

ORACLE RDBMS

Database Administrator's Guide
Version 6.0

ORACLE®

ORACLE DATABASE ADMINISTRATOR'S GUIDE

ORACLE RDBMS VERSION 6.0

ORACLE®
The Relational Database Management System

ORACLE Database Administrator's Guide
Version 6.0

Part No. 3601 - V6.0 November 1988 (Revised April 1989)

Contributing Author: Shelly Dimmick

Contributors: Sanjay Bulchandani, Dwight Cheu, Dennis Cochran,
Elly Faden, Tom Grayson, Chris Harmon, Mike Hartstein, Forrest
Howard, Derry Kabcenell, Ken Jacobs, Andy Laursen, David Martin,
Andrew Mendelsohn, Suzanne Mitchell, Mark Moore, Stephen
Ruppenthal, Kevin Wharton, Mary Winslow.

Copyright © Oracle Corporation 1987, 1989

All rights reserved. Printed in the U.S.A.

Restricted Rights Legend

Use, duplication, or disclosure of the Programs is subject to
restrictions stated in your contract with Oracle Corporation.

Use, duplication, or disclosure of the Programs by the Government is
subject to restrictions for commercial computer software and the
Programs shall be deemed to be licensed with Restricted Rights under
Federal Law.

The information contained in this document is subject to change
without notice. If you find any problems in the documentation, please
report them to us in writing. Oracle Corporation does not warrant
that this document is error-free.

ORACLE, Easy*SQL, Pro*Ada, Pro*COBOL, Pro*Fortran, Pro*PL/I,
Pro*Pascal, SQL*Calc, SQL*Forms, SQL*Graph, SQL*Loader,
SQL*Menu, SQL*Net, and SQL*Plus are registered trademarks of
Oracle Corporation.

Pro*C, SQL*DBA, and SQL*QMX are trademarks of Oracle
Corporation.

PREFACE

This manual describes features and maintenance options of the ORACLE Relational Database Management System (RDBMS). Its contents are intended to help readers understand in detail how ORACLE works and be able to fine tune a system for best performance or specific site requirements. Information is presented in both tutorial and reference fashion.

Information in this manual is applicable to Version 6 of the ORACLE RDBMS with and without the transaction processing option running on all operating systems, with the following caveats:

- Where ORACLE operates the same with or without the transaction processing option it is referred to as ORACLE, Version 6, V6, or the ORACLE RDBMS.
- Some information applies only to ORACLE Version 6 *with* the transaction processing option or only to ORACLE V6 *without* the transaction processing option. These sections are so noted.
- Some information varies depending on the host operating system running the ORACLE RDBMS. These sections are also noted, and you are referred to the *Installation and User's Guide* written for a particular operating system, as in the *Installation and User's Guide for IBM MVS* or the *Installation and User's Guide for DEC VAX/VMS*.
- Some information may apply only to multiple-user ORACLE systems and not to single-user systems (PC/MS-DOS systems). These sections are noted so that users of single-user ORACLE can skip over them.

Audience

This manual is written for persons responsible for the operation of an ORACLE RDBMS. These persons, referred to as *Database Administrators* or *DBAs*, are assumed to be responsible for assuring the smooth ongoing operation of an ORACLE RDBMS and for monitoring its use. The various responsibilities of DBA are described in Chapter 2. In general, the DBA role is to help users use ORACLE and ORACLE-based applications optimally.

If Your Primary Interest is Installation

DBAs are frequently involved in the installation of ORACLE systems; though this manual is not an installation manual, it does contain information relevant to options available during installation. If your interest is installation, you should refer primarily to the *Installation and User's Guide* for your particular operating system, and to the following sections of this manual.

- Chapter 1 What is ORACLE?
- Chapter 3 ORACLE RDBMS File Structure
- Chapter 13 Initial Database Creation
- Chapter 14 Database Startup and Shutdown
- Appendix A Summary of Changes in V6.0
- Appendix B The SQL*DBA Reference
- Appendix D The INIT.ORA Parameters

For information about upgrading between versions of ORACLE, also refer to your *Installation and User's Guide*.

If You are an Application Designer

In addition to DBAs, experienced users of ORACLE and advanced application designers will also find information in this manual useful. For example, these readers may find useful information in the following chapters:

- Chapter 10 SQL Statement Processing
- Chapter 11 Transaction Control
- Chapter 12 Consistency and Concurrency
- Chapter 19 Improving Performance of Applications

Knowledge Assumed of the Reader

Readers of this manual are assumed to be somewhat familiar with ORACLE RDBMS concepts and have some experience using an ORACLE database. Though the early chapters are introductory in nature, following chapters go into much finer detail.

Readers of this manual are also assumed to be familiar with the operating system environment under which they are running ORACLE.

Summary of Changes in Version 6

The following is a very brief list of the changes found in ORACLE RDBMS Version 6 over previous versions. This list is not exhaustive. See Appendix A for more details on the changes.

- **Row-level locking** is the default mode for ORACLE Version 6 with the transaction processing option. In Version 6 without the transaction processing option, locking is the same as in Version 5.
- ORACLE with the transaction processing option provides an interface to **PL/SQL**, Oracle's new procedural extensions to SQL.
- ORACLE includes many **online transaction processing** features such as **deferred writing at commit** and **improved logging**.
- The **new SQL*DBA utility** supports DBA maintenance tasks such as database creation, startup and shutdown, recovery, backup, and system monitoring.
- Numerous changes in database structures give DBAs greatly **increased control over space usage** on a user or object basis.
- The database may be **backed up while it is running**.
- **Portions of a database may be offline** while the rest of the database is online and accessible.
- Users can define **sequences** to generate unique keys.
- The **data dictionary** has been restructured and simplified. Both the new and old data dictionaries are supported.
- **National Language Support** includes new parameters and functions.
- Export and Import now have **incremental and cumulative** options.
- ODL, the Oracle Data Loader, has been replaced by **SQL*Loader**.

Changes in SQL

The Oracle implementation of the SQL language has several changes in V6. Refer to the *SQL Language Reference Manual* for a complete description of any SQL statement. Some of the changes are noted below:

- New SQL commands exist for creating, altering, and dropping databases, sequences, rollback segments, and tablespaces.
- COMMIT and ROLLBACK are now SQL statements.
- The SQL statement SAVEPOINT has been added.
- Implicit rollbacks and execution errors result in statement level rollback, rather than transaction level rollback.
- You can enable multi-statement read consistency using the SET TRANSACTION READ ONLY statement.
- Several commands have been modified, in particular, relating to storage specifications and lock modes.
- Datatypes VARCHAR and CHAR are treated identically; data specified as CHAR is saved in the data dictionary as VARCHAR.
- Due to changes in database structure, all commands referencing partitions or space definitions are obsolete.
- The new CREATE SEQUENCE statement can be used to define sequences, which automatically generate a series of unique ascending or descending integers. The pseudo-columns CURRVAL and NEXTVAL are used to examine and generate key values.
- Referential integrity syntax is supported but not enforced. That is, the additional information is stored in the data dictionary but is not yet used for data validation. See the *SQL Language Reference Manual* for details.
- New functions have been added for National Language Support (CONVERT and REPLACE).
- Nulls sort highest on ascending sequences and lowest on descending sequences.

How this Manual Is Organized

This manual is divided into four parts and several chapters, as described below.

Part I Introduction to ORACLE Version 6

Chapters 1 and 2 discuss the ORACLE line of products and responsibilities of a DBA.

Chapter 1 What is ORACLE?

This chapter introduces the ORACLE products that form the ORACLE RDBMS. It briefly addresses the SQL language on which ORACLE is based and introduces basic terminology necessary for understanding the remainder of the book.

Chapter 2 The Role of the Database Administrator (DBA)

This chapter outlines the responsibilities and concerns of the Database Administrator — the person responsible for maintaining a database. The two DBA users automatically created by ORACLE, SYS and SYSTEM, are described.

Part II ORACLE RDBMS Concepts

Chapters 3 through 12 cover the various features of the RDBMS, such as the structure of the database, database objects, and database processing.

Chapter 3 ORACLE RDBMS File Structure

This chapter discusses the physical operating system files required by an ORACLE RDBMS. Several types of files and their functions are described: ORACLE program files, control files, database files, and redo log files.

Chapter 4 Tablespaces and Segments

This chapter discusses the major logical storage structures and objects of a database: tablespaces and segments. The SYSTEM tablespace and the various types of segments are described: data segments, index segments, rollback segments, and temporary segments.

Chapter 5 User Database Objects

This chapter describes how tables, clusters, views and indexes are stored in an ORACLE database and how space is used throughout database transactions. It also describes block and row formats and the SEQUENCE generator.

Chapter 6 ORACLE Datatypes

This chapter describes the ORACLE datatypes, such as CHAR, NUMBER, and DATE. It describes how they are stored, their limits,

datatype conversion, and how they map to other standard datatypes. This chapter also discusses ROWIDs.

Chapter 7 The Data Dictionary

This chapter describes the set of reference tables known as the data dictionary. To see the actual definitions of the data dictionary tables and views, see Appendix E.

Chapter 8 ORACLE Memory Structures

This chapter describes the memory structures used by an ORACLE RDBMS: the System Global Area (SGA), context areas, and the process global areas.

Chapter 9 ORACLE Process Structure

This chapter discusses various processes active during the operation of an ORACLE database system. It describes single-user and multiple-user ORACLE systems, the five background processes used by multiple user systems, and the difference between two-task and single-task ORACLE systems.

Chapter 10 SQL Statement Processing

This chapter explains what happens when you enter a SQL statement, from creating a cursor, parsing, binding and describing, to executing, fetching, and re-executing the statement. Issues particular to queries, DDL statements, and DML statements are mentioned.

Chapter 11 Transaction Control

This chapter defines the concept of transactions or logical units of work, and the SQL statements (COMMIT, ROLLBACK, and SAVEPOINT) used to control transactions.

Chapter 12 Consistency and Concurrency

This chapter discusses issues related to many users sharing data. It describes read consistency, deadlock detection, and the locks ORACLE uses to guarantee that all operations of multiple transactions do not interfere with each other.

Part III Performing DBA Functions

Chapters 13 through 22 contain instructions for DBA activities, including command options and syntax.

Chapter 13 Initial Database Creation

This chapter briefly describes the steps for creating a database in preparation for its use. This information is intended to summarize, not substitute for, information found in the *Installation and User's Guides*.

Chapter 14 Database Startup and Shutdown

This chapter describes using the SQL*DBA program to start and stop a

database system (an instance and a database). It describes the two commands STARTUP and SHUTDOWN, and their various options.

Chapter 15 Database Backup and Recovery

This chapter describes the files and structures used for database recovery: the redo log files and the rollback segments. It explains decisions the DBA must make and actions he must take to enable or maintain recovery. Media and software failure are distinguished and instructions are given for recovering from these situations.

Chapter 16 Space Management

This chapter describes altering (renaming, enlarging, and adding) the various storage structures of a database (database files, tablespaces, and storage parameters). It expands on the choices you have when using the SQL statements to accomplish these tasks, and makes suggestions for optimizing database storage.

Chapter 17 Security: Database Access

This chapter describes what control the DBA may exercise over granting access to the database to users. It describes the rights and privileges the DBA may GRANT users, setting quotas in tablespaces, and changing passwords. This chapter also contains a checklist of steps for enrolling a new user in the database.

Chapter 18 Security: Database Objects

This chapter addresses controls that users and the DBA have over access to their objects (primarily tables and views) in the database. It describes giving and revoking access, and monitoring it with the ORACLE auditing features. It also describes enabling system auditing and setting system-wide auditing defaults.

Chapter 19 Improving Performance of Applications

This chapter describes ways applications designers can tune their applications for maximum execution efficiency. It describes how the optimizer chooses among access paths, making the best use of indexes, and includes several tuning hints.

Chapter 20 Database Tuning

This chapter describes system-wide actions the DBA can take to tune a database system. Such actions include tuning the INIT.ORA parameters, tablespace management, and certain installation decisions.

Chapter 21 Shared Disk Database Systems

This chapter describes issues unique to shared disk systems (that is, systems made up of multiple instances accessing a common database). An example of these systems is ORACLE running on a DEC VAXCluster.

Chapter 22 Distributed Databases and Distributed Processing

This chapter defines the terms and concepts unique to distributed databases and distributed processing, and then describes decisions faced by DBAs responsible for such systems along with alternative solutions. This chapter should be read in conjunction with other SQL*Net documentation.

Part IV Reference

The appendixes and glossary contain material in reference format to complement information found in the other parts of the manual.

Appendix A Summary of Changes in V6.0

This appendix provides more detail than offered in this preface about the new features in Version 6.0 of the ORACLE RDBMS.

Appendix B The SQL*DBA Reference

This appendix provides a complete reference for each SQL*DBA command (commands which are valid for the DBA utility SQL*DBA).

Appendix C SQL*DBA Errors and Messages

This appendix lists error messages that may be returned by SQL*DBA and suggests possible causes of the error and steps to resolve the error.

Appendix D The INIT.ORA Parameters

This appendix describes the purpose of the INIT.ORA file and its parameters, and lists the INIT.ORA parameters in alphabetical order, with a detailed description of each.

Appendix E Data Dictionary Tables

This appendix lists the data dictionary tables and views in alphabetical order, with descriptions of each column.

Appendix F National Language Support

This appendix describes the effects of varying the values for the INIT.ORA parameter LANGUAGE, in order to support multiple national language conventions.

Appendix G SQL Statement Reference

This appendix contains the reference sections taken verbatim from the *SQL Language Reference Manual* for the SQL statements most often used by DBAs.

Appendix H Database Limits

This appendix is a summary of limits (such as the maximum number of columns per table) on parts and features of an ORACLE database system.

Glossary

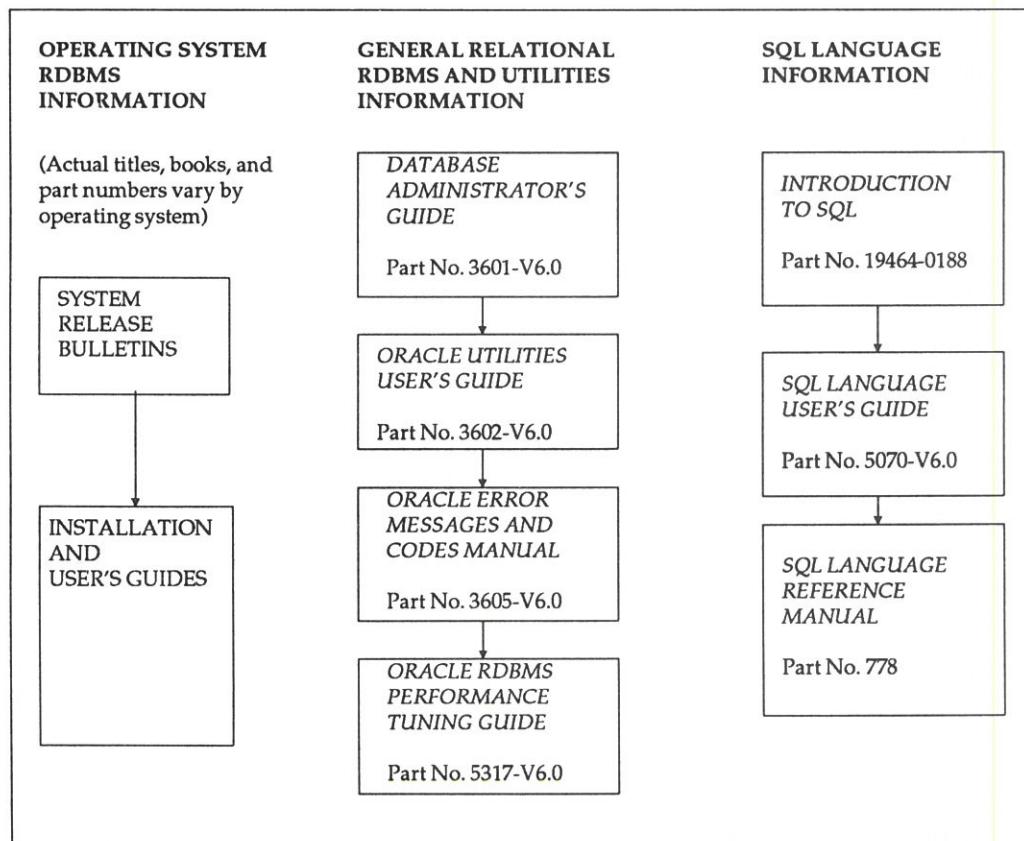
A glossary of terms used in this manual appears after the appendixes.

Related Publications

Because this manual contains information applicable to ORACLE running under any operating system, you will also need to refer to the guides containing information specific to your operating system (typically *System Release Bulletins* or *Installation and User's Guides*). For example:

- *ORACLE for IBM VM/SP Installation and User's Guide*, Oracle Part No. 1003
- *ORACLE for DEC VAX/VMS Installation and User's Guide*, Oracle Part No. 1001

The following diagram shows the primary set of books relating to installing and administering an ORACLE database system.



Conventions Used in this Manual

The following conventions are observed in this manual.

Syntax	appears in the following font for SQL*DBA commands and SQL statements:
	<code>SELECT * FROM EMP</code>
Upper case	indicates a word or phrase to be entered exactly as <i>spelled</i> , although your operating system may allow the words to be entered in lower or mixed case. See your <i>Installation and User's Guide</i> to determine whether your operating system is case-sensitive. The following appear in upper case: <ul style="list-style-type: none">• commands• SQL reserved words and keywords• table and column names used in examples• example filenames. In general, SQL and SQL*DBA commands may be entered in either upper or lower case.
Lower case	indicates a variable where you should substitute an appropriate value, such as the name of a database file.
Square brackets []	indicate optional parameters or keywords.
Curly braces { }	indicate a sequence of parameters or keywords, of which one must be chosen.
Vertical lines	separate optional or mandatory choices.
Ellipses ...	indicate that the preceding item can repeat. You may enter an arbitrary number of similar items.
Other punctuation	such as commas and parentheses should be entered as shown.
File names	appear in capital letters, as in INIT.ORA. Portions of file names that may vary appear in lower case, as in SGADEFx.ORA. Some operating systems treat file names in a case sensitive manner; see your <i>Installation and User's Guide</i> if you require additional information. File extensions or types are notated by a period preceding the type or extension (as in .ORA above); your operating system may use some other separator.

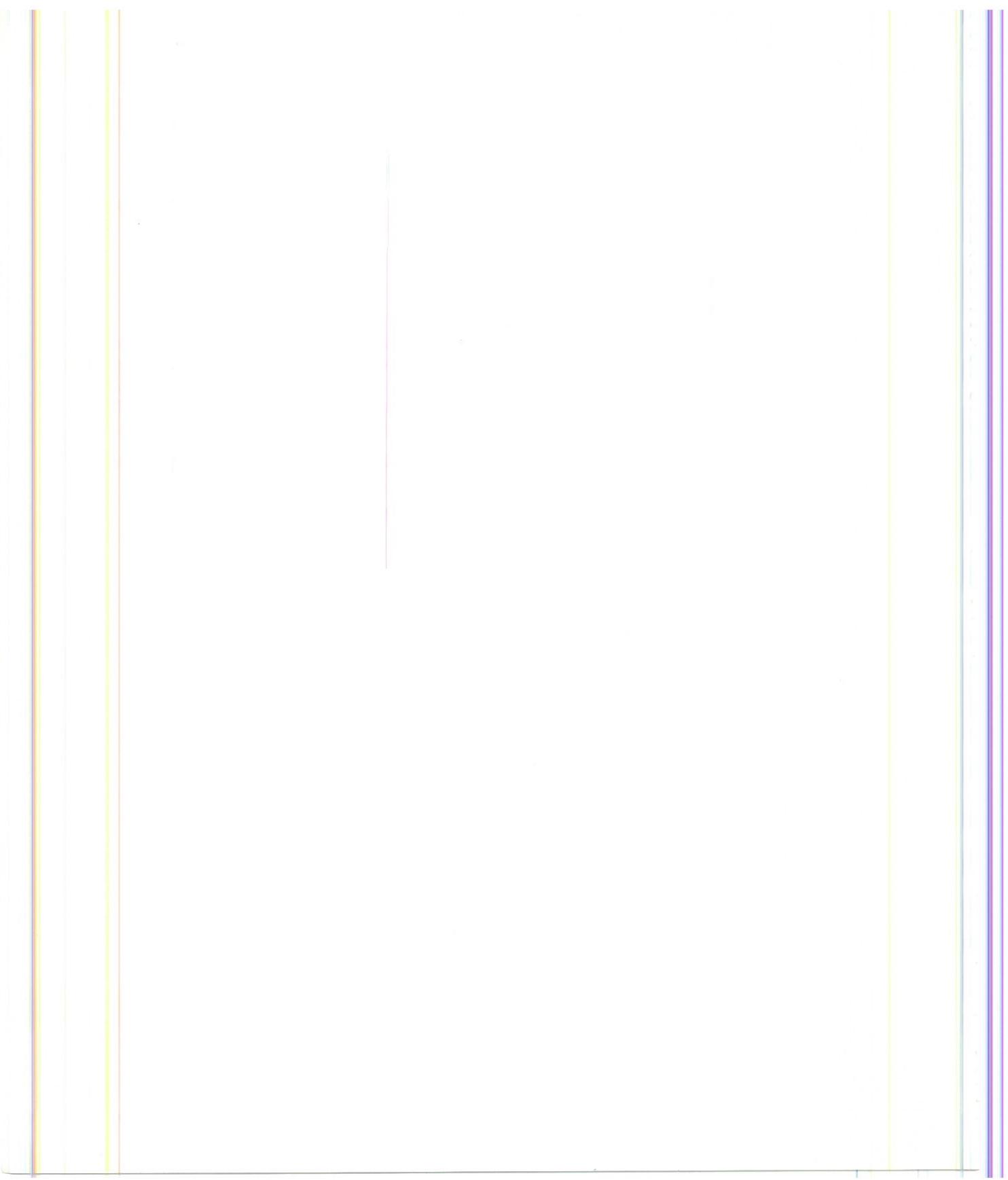
SQL*DBA commands	SQL*DBA commands are shown prefixed by the SQL*DBA prompt to distinguish them from SQL statements, as in: SQLDBA> MONITOR USERS
SQL statements	SQL statements are shown with no prompt, as in: SELECT * FROM EMP

Your Comments are Welcome

We value and appreciate your comments as an ORACLE user and reader of the manuals. As we write, revise, and evaluate, your opinions are the most important input we receive. At the back of this manual is a Reader's Comment Form which we encourage you to use to tell us what you like and dislike about this (or other) ORACLE manuals.

If the form has been used or you would like to contact us, please use the following address or call us at (415) 598-8000.

ORACLE RDBMS Product Manager
Oracle Corporation
20 Davis Drive
Belmont, California 94002



CONTENTS

Chapter 1	What is ORACLE?	1 - 1
	Why Relational?	1 - 2
	Why Choose the ORACLE RDBMS?	1 - 2
	What is the Transaction Processing Option?	1 - 2
	The SQL Language	1 - 3
	Queries	1 - 5
	Data Manipulation Statements (DML)	1 - 5
	Data Definition Statements (DDL)	1 - 5
	Data Control Statements (DCL)	1 - 6
	PL/SQL: A SQL Extension	1 - 6
	The ORACLE Relational Database Management System	1 - 7
	Other ORACLE Products	1 - 8
	Identifying ORACLE Software Versions	1 - 8
	Version Number	1 - 8
	Maintenance Release	1 - 8
	Revision Level	1 - 8
	Port Specific Maintenance Release	1 - 8
	Port Specific Revision Level	1 - 8
	What is an ORACLE Database System?	1 - 9
	What is an ORACLE Database?	1 - 10
	What is an ORACLE Instance?	1 - 11
	Shared Disk Database Systems	1 - 12
	Distributed Database Systems	1 - 13
Chapter 2	The Role of the Database Administrator (DBA)	2 - 1
	Who is the Database Administrator?	2 - 2
	Primary Responsibilities of the DBA	2 - 3
	The DBA's Operating System Account	2 - 3

The Standard DBA Usernames	2 - 4
Username SYS	2 - 4
Username SYSTEM	2 - 5
Connecting as INTERNAL	2 - 5
DBA Privileges	2 - 6
DBA Tools and the SQL*DBA Program	2 - 6
 Chapter 3	
ORACLE RDBMS File Structure	3 - 1
Physical vs. Logical Structures	3 - 3
ORACLE RDBMS Program Files	3 - 3
Database Files	3 - 4
Number of Database Files	3 - 4
Size of Database Files	3 - 5
Blocksize of the Database File	3 - 5
Online and Offline Database Files	3 - 6
Removing Database Files	3 - 6
Control Files	3 - 6
Control File Contents	3 - 7
Number of Control Files	3 - 7
Names of Control Files	3 - 7
Size of Control Files	3 - 7
Redo Log Files	3 - 8
How the Redo Log is Written	3 - 8
Redo Log File Contents	3 - 9
Choosing the Log's Mode of Use	3 - 9
Using the Redo Log in NOARCHIVELOG Mode	3 - 9
Using the Redo Log in ARCHIVELOG Mode	3 - 10
Configuring Redo Log Files	3 - 11
Size of Redo Log Files	3 - 11
Number of Redo Log Files	3 - 11
Checkpoints	3 - 12
 Chapter 4	
Tablespaces and Segments	4 - 1
Tablespaces	4 - 3
The Relationship between Database Files and Tablespaces	4 - 3
The SYSTEM Tablespace	4 - 3
Using Multiple Tablespaces	4 - 4
Associating Users with Tablespaces	4 - 4
Directing Tables to Tablespaces	4 - 5
Online and Offline Tablespaces	4 - 6
Offline Tablespaces	4 - 6
"Accessing" Offline Data	4 - 6

Recovering Offline Tablespaces	4 - 7
Adding New Files or Tablespaces	4 - 7
Segments and Extents	4 - 8
Segment Storage Parameters	4 - 8
Which Storage Parameters are in Effect?	4 - 11
How are Extents Allocated?	4 - 11
When are Extents Deallocated?	4 - 11
Data and Index Segments	4 - 12
Rollback Segments	4 - 12
The Rollback Segment SYSTEM	4 - 13
Multiple Rollback Segments	4 - 13
Contents of the Rollback Segment	4 - 14
How Segments are Allocated to Active Transactions	4 - 14
When Rollback Information is Required	4 - 15
Size of Rollback Segments	4 - 15
Private and Public Rollback Segments	4 - 16
Public Rollback Segments	4 - 16
Private Rollback Segments	4 - 17
Deferred Rollback Segments	4 - 17
Temporary Segments	4 - 18
Operations Requiring Temporary Segments	4 - 18
How Temporary Segments are Allocated	4 - 18
The Bootstrap Segment	4 - 19
 Chapter 5	
User Database Objects	5 - 1
Tables	5 - 2
Who Can Create Tables?	5 - 2
How Tables are Stored	5 - 2
Data Block Format	5 - 3
When Can a Block be Used for Inserts and Updates?	5 - 4
Specifying PCTFREE	5 - 5
Specifying PCTUSED	5 - 6
Choosing Values for PCTUSED and PCTFREE	5 - 7
Row Format	5 - 9
ROWIDs of Rows	5 - 10
Views	5 - 10
Indexes	5 - 11
Indexes, Keys, and Foreign Keys	5 - 12
Unique and Non-Unique Indexes	5 - 12
Concatenated Indexes	5 - 13
When Should I Create an Index?	5 - 13
Which Columns Should I Index?	5 - 14
Creating Indexes	5 - 14
Format of Index Blocks	5 - 15

How Many Indexes Should a Table Have?	5 - 15
How Should I Order Index Columns?	5 - 15
In Which Tablespace Should I Create Indexes?	5 - 16
The Internal Structure of Indexes	5 - 17
Validating Indexes	5 - 18
How Indexes Treat Nulls	5 - 19
Clusters	5 - 19
The Cluster Key	5 - 20
The Cluster Index	5 - 20
Creating Clusters	5 - 21
Choosing Tables to Cluster	5 - 21
Choosing Columns to Cluster	5 - 21
Choosing a CLUSTER SIZE	5 - 22
Choosing PCTUSED and PCTFREE	5 - 22
The SEQUENCE Generator	5 - 23
Listing Sequences	5 - 23
Creating a Sequence	5 - 24
Generating Sequence Numbers with NEXTVAL	5 - 24
Using Sequence Numbers with CURRVAL	5 - 25
Restrictions on NEXTVAL and CURRVAL	5 - 25
Other Sequence Commands	5 - 26
INIT.ORA Parameters Affecting Sequences	5 - 26
Chapter 6	
ORACLE Datatypes	6 - 1
The CHAR and VARCHAR Datatypes	6 - 2
NUMBER Datatype	6 - 2
Using Scale and Precision	6 - 3
Internal Numeric Format	6 - 3
DATE Datatype	6 - 4
Using Julian Dates	6 - 5
LONG Datatype	6 - 5
Restrictions on LONG Data	6 - 6
RAW and LONG RAW Datatypes	6 - 6
How NULL Values are Stored	6 - 7
DB2 and SQL/DS Datatypes	6 - 7
ANSI Datatypes	6 - 8
The Pseudo-Datatype ROWID	6 - 8
Why Use ROWIDs?	6 - 9
Examples of Using ROWID	6 - 9
Data Conversion	6 - 10

Chapter 7	The Data Dictionary	7 - 1
	What is the Data Dictionary?	7 - 2
	The View DICTIONARY	7 - 3
	Access to the Data Dictionary	7 - 3
	The Data Dictionary Tables and Views	7 - 3
	The Data Dictionary Structure	7 - 4
	Views with the Prefix USER	7 - 4
	View with the Prefix ALL	7 - 4
	Views with the Prefix DBA	7 - 5
	ANSI Compatible Views	7 - 5
	Other Views	7 - 5
	How is the Data Dictionary Created?	7 - 8
	How is the Data Dictionary Used?	7 - 9
	Caching of the Data Dictionary for Fast Access	7 - 9
	Other Programs and the Data Dictionary	7 - 9
	Adding New Data Dictionary Items	7 - 9
	Deleting Data Dictionary Items	7 - 10
	Public Synonyms for Data Dictionary Views	7 - 10
	The Dynamic Performance Tables	7 - 10
Chapter 8	ORACLE Memory Structures	8 - 1
	How ORACLE Uses Memory	8 - 2
	Software Areas	8 - 2
	System Global Area (SGA)	8 - 3
	Program Global Area (PGA)	8 - 4
	Context Areas	8 - 5
	Number of Cursors	8 - 5
	Cursor Size	8 - 5
Chapter 9	ORACLE Process Structure	9 - 1
	What is a Process?	9 - 2
	Types of ORACLE Processes	9 - 2
	Single-Task and Two-Task ORACLE	9 - 3
	User Processes	9 - 3
	Single -Task ORACLE	9 - 3
	Two-Task ORACLE and Shadow Processes	9 - 4
	Multiple-Process and Single-Process Instances	9 - 6
	Single-Process ORACLE	9 - 6
	Multiple-Process ORACLE	9 - 6
	The ORACLE Background Processes	9 - 6
	Database Writer (DBWR)	9 - 7
	Log Writer (LGWR)	9 - 7
	System Monitor (SMON)	9 - 8

	Process Monitor (PMON)	9 - 8
	Archiver (ARCH)	9 - 8
	The Program Interface	9 - 8
	Program Interface Structure	9 - 9
	The Program Interface Drivers	9 - 9
	Operating System Communications Software	9 - 9
Chapter 10	SQL Statement Processing	10 - 1
	What Constitutes a SQL Statement?	10 - 2
	Cursors and Context Areas	10 - 2
	A Simple Example of SQL Statement Execution	10 - 3
	Using a Cursor to Re-execute Statements	10 - 4
	Statement Processing in More Detail	10 - 4
	Opening and Closing Cursors	10 - 5
	Parsing Statements	10 - 6
	Binding Variables	10 - 7
	Describing Results	10 - 7
	Defining Output	10 - 7
	Execution	10 - 7
	DDL Processing	10 - 8
	Query Processing	10 - 8
	Fetching Rows of a Query Result	10 - 8
Chapter 11	Transaction Control	11 - 1
	Definition of a Transaction	11 - 2
	The Significance of Transactions	11 - 3
	Starting and Ending a Transaction	11 - 3
	Read-Only Transactions	11 - 4
	How are Transactions Defined in ORACLE Utilities?	11 - 4
	Controlling Transactions with SQL Statements	11 - 5
	The COMMIT WORK Statement	11 - 5
	What Happens When You Commit?	11 - 6
	Using SAVEPOINTS to Save Work	11 - 6
	Statement Level Rollback	11 - 8
	The ROLLBACK WORK Statement	11 - 8
	Invoking COMMIT or ROLLBACK	11 - 9
	Explicit Commits/Rollbacks	11 - 9
	Implicit (Automatic) Commits/Rollbacks	11 - 9

Consistency and Concurrency	12 - 1
General Concurrency Issues	12 - 2
Read Consistency	12 - 3
Read Consistency Examples	12 - 4
The "Snapshot Too Old" Message	12 - 4
Introduction to Locking	12 - 5
What Resources can be Locked?	12 - 5
Duration of Locks	12 - 6
Who can Explicitly Lock?	12 - 6
Default vs. Manual Locking	12 - 6
Types of Locks	12 - 7
Data or DML Locks	12 - 8
Data Locking Behavior for ORACLE with the Transaction	
Processing Option	12 - 8
Read Consistent Statements for Queries, Updates, and Deletes	12 - 8
Default Locking for Queries	12 - 9
Default Locking for Updates and Deletes	12 - 10
Overriding Default Locking	12 - 11
Using the LOCK TABLE Statement	12 - 12
Shared and Exclusive Locks	12 - 12
Descriptions of Lock Modes	12 - 13
Lock Escalation	12 - 14
Lock Conversion	12 - 14
When to Use SELECT FOR UPDATE	12 - 14
Using SET TRANSACTION READ ONLY	12 - 15
Dictionary Locks	12 - 16
Parse Locks	12 - 16
DDL Locks	12 - 16
Dictionary Locks and Clusters	12 - 16
Internal Locks and Latches	12 - 17
Latches	12 - 17
Internal Locks	12 - 17
Dictionary Cache Locks	12 - 17
File and Log Management Locks	12 - 17
Tablespace and Rollback Segment Locks	12 - 17
Monitoring Locks with SQL*DBA	12 - 18
Deadlock Detection	12 - 19
Avoiding Deadlocks	12 - 20
The INIT.ORA Parameters SERIALIZABLE and ROW_LOCKING	12 - 20
Setting SERIALIZABLE to TRUE	12 - 21
Setting ROW_LOCKING to INTENT	12 - 21
Summary of Non-Default Locking Options	12 - 21

Chapter 13	Initial Database Creation	13 - 1
	What is Database Creation?	13 - 2
	The Steps for Creating a Database	13 - 2
	What Does CREATE DATABASE Do?	13 - 6
	After Creating a Database	13 - 7
Chapter 14	Database Startup and Shutdown	14 - 1
	Startup and Shutdown Concepts	14 - 2
	Mounted vs. Open Databases	14 - 2
	Dismounted	14 - 3
	Mounted, but not Open	14 - 3
	Open	14 - 3
	The SQL*DBA Program	14 - 4
	Access to SQL*DBA	14 - 4
	Starting an ORACLE Instance and Database	14 - 5
	Startup with Open Database	14 - 5
	Mounting and Opening in Separate Steps	14 - 6
	Startup for DBAs Only	14 - 6
	Startup with FORCE Option	14 - 7
	Startup with Tablespaces Offline	14 - 7
	Automatic Startup with Operating System Start	14 - 7
	Starting Remote Instances	14 - 7
	The INIT.ORA Parameter File	14 - 8
	Using Alternate Names for INIT.ORA	14 - 8
	Sample INIT.ORA File	14 - 8
	Groups of INIT.ORA Parameters	14 - 9
	Why Change Parameters?	14 - 9
	Shutting Down an ORACLE Instance and Database	14 - 10
	Normal Shutdown	14 - 10
	Shutdown IMMEDIATE	14 - 10
	Shutdown ABORT	14 - 11

Database Backup and Recovery	15 - 1
Why is Recovery Important?	15 - 2
ORACLE Recovery Features	15 - 3
Types of Failure Requiring Recovery	15 - 3
User Error	15 - 3
Statement Failure	15 - 3
Process Failure	15 - 4
Instance Failure	15 - 4
Media Failure	15 - 4
Basic Recovery Steps	15 - 5
Rolling Forward with the Redo Log	15 - 6
Rolling Back with the Rollback Segments	15 - 7
Instance Recovery	15 - 7
Media Recovery	15 - 8
Database Restoration	15 - 9
How the Redo Log Works	15 - 9
Using the Log in ARCHIVELOG Mode	15 - 10
The Online Redo Log	15 - 10
The Offline Redo Log	15 - 10
Using the Log in NOARCHIVELOG Mode	15 - 11
Checkpoints	15 - 12
Checkpoints for Speeding Recovery	15 - 13
Checkpoints When a Redo Log File Fills	15 - 13
DBA Decisions Regarding Recovery Structures	15 - 13
Creating the Online Redo Log	15 - 13
Choosing the Redo Log Mode	15 - 14
Choosing the Number of Log Files	15 - 14
Setting the Size of the Redo Log Files	15 - 15
Setting Devices for Redo Log Files	15 - 15
Altering the Log	15 - 15
Changing between NOARCHIVELOG and ARCHIVELOG Mode	15 - 16
Choosing a Checkpoint Interval	15 - 16
Creating Rollback Segments	15 - 16
Choosing the Number of Rollback Segments	15 - 17
Setting Storage Allocations for Rollback Segments	15 - 18
Archiving Redo Log Data in ARCHIVELOG Mode	15 - 19
Log Sequence Numbers	15 - 19
Automatic Archiving	15 - 20
Enabling Automatic Archiving via INIT.ORA	15 - 20
Enabling Automatic Archiving via SQL*DBA	15 - 20
Setting the Archive Destination	15 - 20
Disabling Automatic Archiving	15 - 21
Manual Archiving	15 - 21

Performing Backups	15 - 22
Backing Up a Closed Database	15 - 23
Backing Up an Open Database	15 - 23
Media Recovery Procedures	15 - 25
Recovering a Database File not in the Tablespace SYSTEM	15 - 25
Tablespace Recovery	15 - 25
Database Recovery	15 - 27
Interrupting Recovery	15 - 27
Resuming Recovery	15 - 27
Recovering a Database File in the Tablespace SYSTEM	15 - 28
Recovering an Online Redo Log File	15 - 28
Recovering Control Files	15 - 28
Point in Time Recovery	15 - 29
Export/Import	15 - 30
Export	15 - 30
Full Database Exports	15 - 31
Incremental Exports	15 - 32
Cumulative Exports	15 - 32
Import	15 - 32
Using IMPORT for Full Database Recovery	15 - 32
Chapter 16	
Space Management	16 - 1
Database File Maintenance	16 - 2
Redo Log File Maintenance	16 - 2
Adding New Redo Log Files	16 - 3
Dropping Redo Log Files	16 - 4
Renaming Redo Log Files	16 - 4
Rollback Segment Maintenance	16 - 5
Creating New Rollback Segments	16 - 5
Public versus Private Rollback Segments	16 - 5
Monitoring Rollback Segments	16 - 6
Changing the Size of Rollback Segments	16 - 6
Dropping Rollback Segments	16 - 6
Tablespace Maintenance	16 - 7
Creating a New Tablespace	16 - 7
Enlarging a Tablespace (Adding a File)	16 - 8
Monitoring Space Usage in a Tablespace	16 - 8
Setting User Quotas for Tablespace Usage	16 - 9
Setting Default Storage Parameters for a Tablespace	16 - 9
Changing Default Storage Parameters	16 - 9
Renaming Files in a Tablespace	16 - 9
Taking a Tablespace Offline	16 - 9

Bringing a Tablespace Online	16 - 10
Dropping Tablespaces	16 - 10
Managing Storage for Tables and Clusters	16 - 11
Locating Tables in Tablespaces	16 - 11
Setting Table Storage Parameters	16 - 12
Calculating Space Required by a Table	16 - 12
Monitoring Actual Space Used	16 - 13
Increasing Table Storage Parameters	16 - 13
Managing Index Storage	16 - 13
Locating Indexes in Tablespaces	16 - 13
Calculating Space Required by an Index	16 - 13
Choosing Storage Parameters for Indexes	16 - 14
Limits on Indexes	16 - 14
Managing Clusters	16 - 15
Creating Clusters	16 - 15
Single-Table Clusters	16 - 15
Choosing the SIZE Parameter for Clusters	16 - 15
Monitoring Space used by Clusters	16 - 16
Loading Clusters	16 - 16
Dropping Clusters	16 - 17
Managing Temporary Segments	16 - 17
Monitoring the Number of Temporary Segments	16 - 17
Monitoring the Location of Temporary Segments	16 - 17
Monitoring the Size of Temporary Segments	16 - 17
Chapter 17	
Security: Database Access	17 - 1
Enrolling Users	17 - 2
Users with CONNECT Privilege	17 - 3
Users with RESOURCE Privilege	17 - 4
Users with DBA Privilege	17 - 4
Assigning Users Default Tablespaces	17 - 5
Setting Tablespace Quotas for Users	17 - 6
Changing Tablespace Quotas	17 - 6
Revoking Access to a Tablespace	17 - 6
Automatic Logins	17 - 7
Summary of Steps for Enrolling Users	17 - 8
Changing Passwords	17 - 9
Dropping Users	17 - 9
Auditing Database Access	17 - 10
Enabling System Auditing	17 - 10
Setting System-wide Auditing Options	17 - 10
Listing System-wide Auditing Settings	17 - 11
Seeing Auditing Results in the Audit Trail	17 - 11
Auditing Connections	17 - 12

	Auditing Resource Use	17 - 12
	Auditing DBA Actions	17 - 13
	Auditing Attempts to Access Non-existent Objects	17 - 13
	Disabling System-wide Auditing	17 - 13
	The Special "User" PUBLIC	17 - 14
	Public Synonyms	17 - 14
Chapter 18	Security: Database Objects	18 - 1
	Access to Table Data	18 - 2
	Access to All Data in a Table	18 - 2
	Access to Views (or Selected Data in a Table)	18 - 2
	Access to UPDATE Columns	18 - 3
	Access to Sequence Numbers	18 - 3
	Revoking Access Privileges	18 - 4
	Auditing Database Objects	18 - 4
	Who Can Use Auditing?	18 - 5
	Setting Table Auditing Options	18 - 5
	Auditing by Session or by Access	18 - 6
	Auditing Views	18 - 6
	Default Table Auditing	18 - 6
	Listing Current Auditing Options	18 - 7
	Seeing Auditing Results in the Audit Trail	18 - 7
	Disabling Table Auditing	18 - 8
	The Function USERENV	18 - 8
Chapter 19	Improving Performance of Applications	19 - 1
	The Importance of Good Relational Table Design	19 - 2
	Operator Precedence	19 - 3
	Using the SQL Language Effectively	19 - 4
	Optimizations Automatically Performed	19 - 5
	Using DISTINCT	19 - 5
	Using GROUP BY	19 - 5
	Subqueries	19 - 5
	Single Row Query Paths	19 - 5
	Writing SQL Statements to Take Advantage of Indexes	19 - 6
	Creating Effective Indexes	19 - 7
	Single Indexes	19 - 9
	Indexes and Null Values	19 - 9
	Multiple Indexes on One Table	19 - 10
	Choosing Among Multiple Indexes	19 - 10
	Suppressing the Use of Indexes	19 - 10
	Optimizing Various SQL Statements	19 - 11
	Optimizing Queries (SELECTS)	19 - 11

Optimizing NOTS	19 - 12
Optimizing ORS	19 - 12
The ORACLE Sort/Merge Routine	19 - 13
Optimizing ORDER BY	19 - 14
Optimizing GROUP BY	19 - 14
Optimizing Joins	19 - 15
Unindexed Joins	19 - 15
Indexed Joins	19 - 15
Array Processing	19 - 18
Avoid Reparsing and Rebinding	19 - 18
Chapter 20	
Database Tuning	20 - 1
Basic Tuning Concepts	20 - 2
Know Your Users' Requirements	20 - 3
How to Tune Effectively	20 - 3
Levels of Performance Tuning	20 - 3
Primary Tuning Features	20 - 4
Tuning at Installation Time	20 - 5
Examples of O/S Tuning Steps	20 - 5
Setting Process Priorities	20 - 5
Avoiding Disk Fragmentation	20 - 5
Quick Tuning Techniques	20 - 6
Optimizing I/O	20 - 6
Monitoring I/O with SQL*DBA	20 - 7
Using Default Tablespaces	20 - 7
Minimizing the Number of Extents	20 - 8
When More Extents are Desirable	20 - 8
Reducing Frequency of Extent Allocations	20 - 9
Separating Table Data from Index Data	20 - 10
"Striping" Large Tables	20 - 10
Database and Control File I/O	20 - 10
Log File I/O	20 - 11
Optimizing the Buffer Manager	20 - 11
Monitoring the Buffer	20 - 12
Increasing the Number of Database Buffers	20 - 12
Reducing Block Level Contention	20 - 13
Tuning the DBWR Process	20 - 13
How DBWR Works	20 - 13
Monitoring DBWR	20 - 14
Minimizing DBWR Activity	20 - 14
Tuning the Redo Log	20 - 16
Summary of Redo Log Optimizations	20 - 16
Monitoring the Redo Log	20 - 17
Optimizing the Log Buffer	20 - 17

Reducing Checkpoint Frequency	20 - 17
Reducing Log Allocation Frequency	20 - 18
The Interaction between Parameters and Statistics	20 - 19
Tuning Rollback Segments	20 - 20
Monitoring Rollback Segments	20 - 21
Read Consistency	20 - 21
Using Multiple Rollback Segments	20 - 21
Choosing a Size for Rollback Segments	20 - 22
Caching Rollback Segments	20 - 22
Monitor Ongoing Database Storage and Use	20 - 23
Choosing Reasonable Auditing Options	20 - 23
Problem Determination	20 - 23
CPU-Bound	20 - 23
Latch-Bound	20 - 23
DBWR-Bound	20 - 24
LGWR-Bound	20 - 24
I/O Bound	20 - 24
Chapter 21	
Shared Disk Database Systems	21 - 1
What is a Shared Disk System?	21 - 2
Using SQL*DBA for Multiple Instances	21 - 2
Startup with SHARED Option	21 - 2
INIT.ORA Parameters Relevant to Shared Disk Systems	21 - 3
Using Rollback Segments	21 - 3
Private Rollback Segments	21 - 4
Public Rollback Segments	21 - 4
Backup and Recovery for Shared Disk Systems	21 - 4
Using the Redo Log	21 - 4
Allocating Log Space to Instances	21 - 4
Checkpoints	21 - 5
Setting the Log's Mode	21 - 5
Archiving	21 - 5
Instance Recovery	21 - 6
Chapter 22	
Distributed Databases and Distributed Processing	22 - 1
What is SQL*Star?	22 - 2
What is Distributed Processing?	22 - 2
What is a Distributed Database?	22 - 3
Benefits of ORACLE Distributed Processing	22 - 3
Location Transparency	22 - 3
Site Autonomy	22 - 4
The Role of Clients and Servers	22 - 5
Connecting Clients and Servers	22 - 5

Division of Labor between Clients and Servers	22 - 5
Connecting between Versions	22 - 5
Features of ORACLE and SQL*Net	22 - 6
Switching between Default and Other Hosts	22 - 7
Deciding Where to Locate Applications	22 - 7
Moving Data with the COPY Command	22 - 8
Setting Up Database Links	22 - 9
Private DBLINKs	22 - 10
PUBLIC DBLINKs	22 - 10
Choosing ORACLE Usernames to Access Remote Data	22 - 10
Using One Central Account for Clients	22 - 10
Individual Accounts for Clients	22 - 11

Appendix A

Summary of Changes in V6.0	A - 1
Database Terminology	A - 2
The SQL*DBA Utility	A - 2
SQL*Loader Replaces ODL	A - 3
Enhanced Data Dictionary	A - 3
Database File Structure	A - 4
Control Files	A - 4
Database Files	A - 4
Redo Log Files	A - 4
INIT.ORA File	A - 4
Database Structures	A - 5
New Structures	A - 5
Changed Structures	A - 6
Transaction Control	A - 7
New Lock Modes	A - 7
Password Encryption	A - 7
Error Codes	A - 7
Changes in SQL	A - 8
Version 5 and Version 6 Terminology	A - 9

Appendix B

The SQL*DBA Reference	B - 1
Invoking SQL*DBA	B - 2
Entering SQL*DBA Commands	B - 2
System Privileged SQL*DBA Commands	B - 3
Entering SQL Commands	B - 3
Entering PL/SQL Commands	B - 4
Running Command Files	B - 4
Entering Commands from the Operating System	B - 4
Using SQL*DBA in a Distributed Environment	B - 4
Monitoring Database Use with MONITOR	B - 5

Dynamic Performance Tables used by MONITOR	B - 6
Granting Other Users Access to MONITOR Screens	B - 6
Characteristics of All Displays	B - 6
The MONITOR PROCESS Display	B - 8
The MONITOR USER Display	B - 9
The MONITOR TABLE Display	B - 10
The MONITOR LOCK Display	B - 11
The MONITOR LATCH Display	B - 13
The MONITOR STATISTICS Display	B - 15
The MONITOR I/O Display	B - 23
The MONITOR ROLLBACK SEGMENT Display	B - 24
The MONITOR FILE I/O Display	B - 25
ARCHIVE LOG	B - 26
CONNECT	B - 28
DISCONNECT	B - 30
EXIT	B - 31
HOST	B - 32
MONITOR	B - 33
RECOVER	B - 35
REMARK	B - 36
SET	B - 37
SHOW	B - 39
SHUTDOWN	B - 41
SPOOL	B - 42
STARTUP	B - 43
Appendix C	
SQL*DBA Error Codes and Messages	C - 1
DBA Errors	C - 2
LCC Errors	C - 7
Appendix D	
THE INIT.ORA PARAMETERS	D - 1
Specifying Parameters in the INIT.ORA File	D - 2
Displaying Current Parameter Values	D - 3
Groups of Parameters	D - 3
Variable Parameters	D - 4
Dictionary Cache Parameters (with Prefix "DC")	D - 4
Global Cache Parameters (with Prefix "GC")	D - 5
Operating System Dependent Parameters	D - 5
Dependent Parameters	D - 5
When Parameters Are Set Incorrectly	D - 5
Individual Parameter Descriptions	D - 6

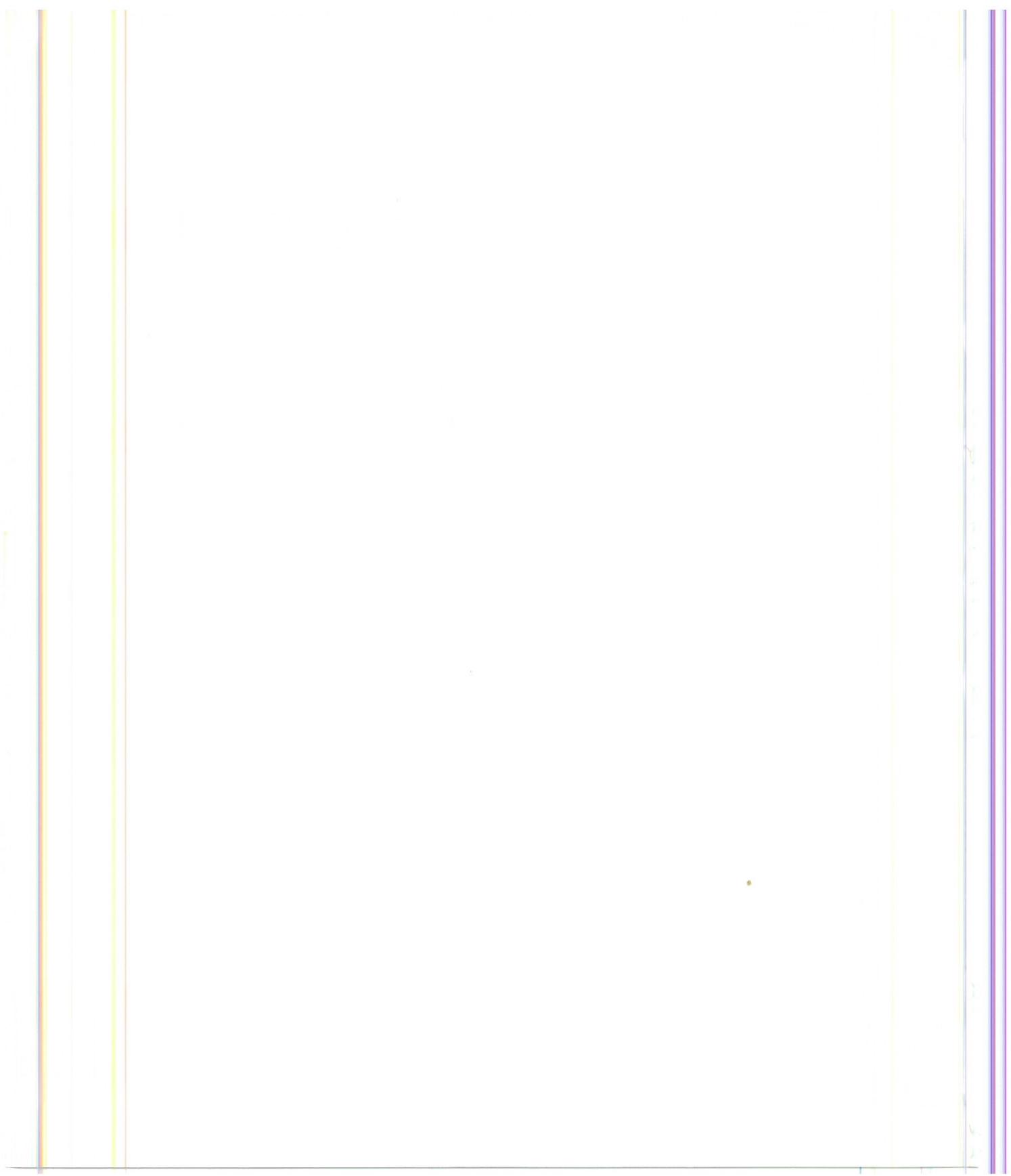
Appendix E	Data Dictionary Tables	E - 1
	The Data Dictionary Views	E - 2
	The Dynamic Performance Tables	E - 29
	Access to the Dynamic Performance Tables	E - 29
	Tables Used by MONITOR Displays	E - 29
	Description of Individual Tables	E - 31
Appendix F	National Language Support	F - 1
	What does National Language Support Provide?	F - 2
	Specifying National Language Operation	F - 2
	Language Used for ORACLE Messages	F - 3
	Language Used for Day and Month Names	F - 4
	Week and Day Number Calculation	F - 5
	Processing Data in Multiple Character Sets	F - 6
	What are Character Sets?	F - 6
	Problems of Handling Multi-lingual Data	F - 6
	Storing Character Data	F - 7
	SQL's Approach to Different Character Sets	F - 7
	Converting Between Upper and Lower Case	F - 7
	Converting Between Character Sets	F - 8
	Using Replacement Characters	F - 8
	The REPLACE Function	F - 9
	Sorting Character Data	F - 10
	Using Non-Default Sorting	F - 10
	Comparing Character Strings	F - 11
	Specifying the LANGUAGE Parameter on a User Basis	F - 11
Appendix G	SQL Statement Reference	G - 1
	ALTER DATABASE	G - 2
	ALTER ROLLBACK SEGMENT	G - 5
	ALTER TABLESPACE	G - 6
	ALTER USER	G - 8
	AUDIT (Form I)	G - 9
	AUDIT (Form II)	G - 11
	CREATE DATABASE	G - 13
	CREATE ROLLBACK SEGMENT	G - 16
	CREATE TABLESPACE	G - 17
	DROP ROLLBACK SEGMENT	G - 19
	DROP TABLESPACE	G - 20
	GRANT (Form I)	G - 21
	GRANT (Form II)	G - 23
	GRANT (Form III)	G - 24
	NOAUDIT (Form I)	G - 27

NOAUDIT (Form II)	G - 28
REVOKE (Form I)	G - 29
REVOKE (Form II)	G - 30
REVOKE (Form III)	G - 31
Storage clause	G - 33
Appendix H	
Database Limits	H - 1
Glossary	
Index	

P A R T

I

INTRODUCTION TO ORACLE VERSION 6



1

WHAT IS ORACLE?

*I am Sir Oracle,
And when I ope my lips, let no dog bark!*

Shakespeare: *The Merchant of Venice*

This chapter introduces the ORACLE Relational Database Management System (RDBMS). It introduces basic terms and concepts to lay a foundation for the rest of the manual. This chapter describes:

- the difference between ORACLE Version 6 with the transaction processing option and without the transaction processing option
- the SQL language, upon which ORACLE tools are based
- how to identify ORACLE software versions
- the definitions of the terms database, instance, and database system, and their components.

Why Relational?

Over the past several years, relational database management systems have become the most widely accepted way to manage data. Relational systems offer benefits such as:

- easy access to all data
- flexibility in data modeling
- reduced data storage and redundancy
- independence of physical storage and logical data design
- a high-level data manipulation language (SQL).

As the technologies associated with relational database management systems have grown rapidly in recent years, the appeal of relational databases has become apparent to a much wider audience.

The phenomenal growth of the relational technology has led to more demand for RDBMSs in environments ranging from personal computers to large, highly secure CPUs, with users ranging from very casual to very sophisticated.

Why Choose the ORACLE RDBMS?

Oracle Corporation was the first company to offer a true relational DBMS commercially, and has continually led innovations in the field of RDBMSs. The Oracle Corporation strategy of offering an RDBMS that is portable, compatible, and connectable results in a very powerful tool for users. You learn basic concepts once, and you can apply those concepts across numerous hardware and software environments.

What is the Transaction Processing Option?

This manual describes two versions of ORACLE software:

- ORACLE RDBMS Version 6.0 with the transaction processing option
- ORACLE RDBMS Version 6.0 without the transaction processing option.

The ORACLE RDBMS is a high-performance, fault-tolerant database management system, especially designed for online transaction processing and large database applications.

ORACLE V6 contains numerous changes in the database and SQL language which make it superior to earlier ORACLE RDBMS versions. These changes include:

- deferred writing at commit to improve transaction performance
- improvements in user and DBA control over space usage
- online backup and recovery, and improved logging
- additional fault tolerance features
- savepoints and statement level rollback
- a new database administration tool, SQL*DBA
- incremental export.

The transaction processing option offers two features which contribute to very high levels of transaction processing throughput:

- the row level lock manager
- PL/SQL, a procedural language extension to SQL.

ORACLE Version 6 with and without the transaction processing option contain the same features with the exception of the row level lock manager and PL/SQL. This manual indicates differences between ORACLE Version 6 with and without the transaction processing option when those differences may be significant to users.

The SQL Language

At the heart of the ORACLE RDBMS is the SQL (pronounced "sequel") data language — an English-like language that is used for most database activities. SQL is simple enough to allow beginning users to access data easily and quickly, yet it is powerful enough to offer programmers all the capability and flexibility they require.

SQL was developed and defined by IBM Research, and has been refined by the American National Standards Institute (ANSI) as the standard language for relational database management systems.

The SQL implemented by Oracle Corporation is a superset of the standard SQL data language. Commands in this book are either SQL statements or SQL*DBA commands.

SQL statements

These are statements designed to work with relational database data. SQL statements (also called commands) may be used in many environments, including ORACLE-supplied products (such as SQL*Forms), utilities (such as SQL*DBA), and user-written programs (such as Pro*COBOL).

The remainder of this section describes the main categories of SQL statements. For a complete SQL language reference, refer to the *SQL Language Reference Manual*. Appendix G contains portions of that guide listing SQL statements often used by DBAs.

SQL*DBA commands

SQL*DBA commands should not be confused with SQL statements. These commands help the DBA perform database maintenance functions and may be used only within the SQL*DBA utility. This book is the primary source for these commands; a reference can be found in Appendix B.

The SQL data language presumes that database data is found in *tables* (such as table EMP). Each table is defined by a *tablename* and a set of *columns*. Each column has a *column name* (such as EMPNO, ENAME, JOB) and a *datatype* (such as CHAR, for character, or NUMBER). Columns can contain null values if they have not been declared NOT NULL. Columns are sometimes called *fields* or *attributes*.

Actual data values are found in *rows* of tables (also called *records* or *tuples*). The following sample rows come from a table often referenced in ORACLE documentation, a table of employee data called EMP.

EMPNO	ENAME	JOB	MGR	HIREDATE	SALARY	COMM	DEPTNO
7329	SMITH	CLERK	7902	17-DEC-80	800	300	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20

SQL language statements are often divided into four categories:

- Queries
- Data Manipulation statements (DML)
- Data Definition statements (DDL)
- Data Control statements (DCL).

These categories are often simplified into two (combining the first two as DML and the last two as DDL). The categories occasionally have different implications on database operation. This book assumes that you understand which SQL statements fall in which category. Each category is briefly described below.

Queries

Queries are statements that retrieve data, in any combination, expression, or order. Queries usually begin with the SQL reserved word SELECT, followed by the data desired, and the tables or views containing the source data, as in:

```
SELECT ENAME, MGR  
FROM EMP;
```

Queries do not change any data; they only retrieve data. They are often categorized with DML statements.

Data Manipulation Statements (DML)

DML statements are used to change the data in one of three ways:

- INSERT new rows of data into a table
- UPDATE column values in existing rows
- DELETE rows from tables.

Some examples of DML statements are:

```
INSERT INTO EMP VALUES (1234, 'DAVIS', 'SALESMAN', 7698,  
'14-FEB-1988' 1600, 500, 30)
```

```
DELETE FROM EMP WHERE NAME IN ('WARD', 'JONES')
```

Along with queries, DML statements are generally the most frequently used statements.

The LOCK, COMMIT WORK, and ROLLBACK WORK statements are also DML statements.

Data Definition Statements (DDL)

DDL statements are used to define and maintain database objects (including database tables) and to drop them when the objects are no longer needed. DDL statements include all CREATE statements (such as CREATE TABLE and CREATE VIEW) and corresponding ALTER and DROP statements.

For example:

```
CREATE TABLE PLANTS (COM_NAME CHAR 15, LAT_NAME CHAR 20)  
DROP TABLE PLANTS
```

Data Control Statements (DCL)

DCL statements are used to control two types of access:

- access to the database (using, for example, GRANT CONNECT)
- access to database data (using, for example, GRANT SELECT or REVOKE DELETE).

DCL statements allow one user to let other users see, change, and use data in his tables, and even to pass their privileges on to other users (GRANT SELECT ... WITH GRANT OPTION). Data control statements also include the AUDIT statements.

DCL and DDL statements are often categorized together; in fact, ORACLE RDBMS treats these statements identically.

PL/SQL: A SQL Extension

ORACLE with the transaction processing option includes a procedural superset of the SQL language, called PL/SQL. PL/SQL extends the SQL language by offering block structured procedural constructs combined with SQL's non-procedural capabilities. PL/SQL's features include:

- variable declarations and assignments
- conditional control (IF, THEN, ELSE)
- looping (FOR, WHILE, EXIT WHEN)
- exception handling.

A principal benefit of PL/SQL is that ORACLE with the transaction processing option can execute an entire procedure with one request, instead of a single SQL statement at a time. ORACLE users of distributed processing applications will particularly benefit because of reduced network traffic.

Documentation for PL/SQL is available in the *PL/SQL User's Guide and Reference*.

The ORACLE Relational Database Management System

The ORACLE RDBMS is the central ORACLE product. It includes the database manager and several tools intended to assist users and DBAs in the maintenance, monitoring, and use of data.

The core of the RDBMS is the *kernel*. The kernel handles the following tasks:

- manages storage and definition of data
- controls and limits data access and concurrency
- allows backup and recovery of data
- interprets SQL and PL/SQL.

One part of the kernel is the *optimizer*. The optimizer examines alternate *access paths* to the data, to find the optimal path to resolve a given query. For example, it looks at the wording of the query to see which indexes will be most helpful. The optimizer does not execute the query.

Several utilities are included in the ORACLE RDBMS product. Utilities primarily for DBAs are documented in this manual.

SQL*DBA	A DBA utility used for several purposes, including starting and stopping a database, monitoring its use, and performing backup and recovery. You will find a complete reference of SQL*DBA commands in Appendix B.
CRT	A utility to define screen display characteristics for full-screen products (like SQL*Forms) and to define or modify keypad function assignments. The CRT utility is documented in the <i>ORACLE Utilities User's Guide</i> .
Export/Import	A utility to move ORACLE data to and from files. Export files can be used to archive data or to move data between ORACLE databases on the same or different operating systems. Some information on Export/Import can be found in Chapter 15; the primary reference is the <i>ORACLE Utilities User's Guide</i> .
SQL*Loader	A user utility to load data into ORACLE RDBMS tables from operating system standard files. The primary reference for SQL*Loader is the <i>ORACLE Utilities User's Guide</i> .

Other ORACLE Products	The ORACLE product line offers a variety of tools and functionality to ORACLE users. Beginning users can use menu-driven or full-screen applications. Sophisticated users can use several tools to access ORACLE data. These tools include spreadsheet applications, interactive interfaces, or programmatic interfaces available for several programming languages.
------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Identifying ORACLE Software Versions

As ORACLE products are always undergoing development and change, several versions of the products are in use at any one time. To fully identify a software product, as many as five numbers may be required.

Version Number	The version number, such as Version 6, is the most general identifier. New versions are released when significant new functionality has been added to a product.
Maintenance Release	The maintenance release number signifies different releases of the general version, starting with 0, as in Version 6.0 or Version 6.1. The maintenance level increases when bug fixes or new features to existing programs are available.
Revision Level	The revision level identifies a specific level of the object code. This number is used primarily by Oracle Corporation to fully identify an ORACLE system. Usually a site will receive only one revision level of a particular product maintenance release (not necessarily revision 1).
Port Specific Maintenance Release	On some operating systems a fourth number may be required to identify a particular maintenance release of a software product on that operating system. This number may be different from the product maintenance release number.
Port Specific Revision Level	On some operating systems a fifth number may be required to identify a particular revision level for the operating system maintenance release.

For example, a tape might be labelled 6.0.20/1.3 as follows:

VERSION NUMBER	MAINTENANCE RELEASE NUMBER	REVISION NUMBER	/ PORT RELEASE NUMBER	REVISION NUMBER
6	.0	.20	/ 1	.3

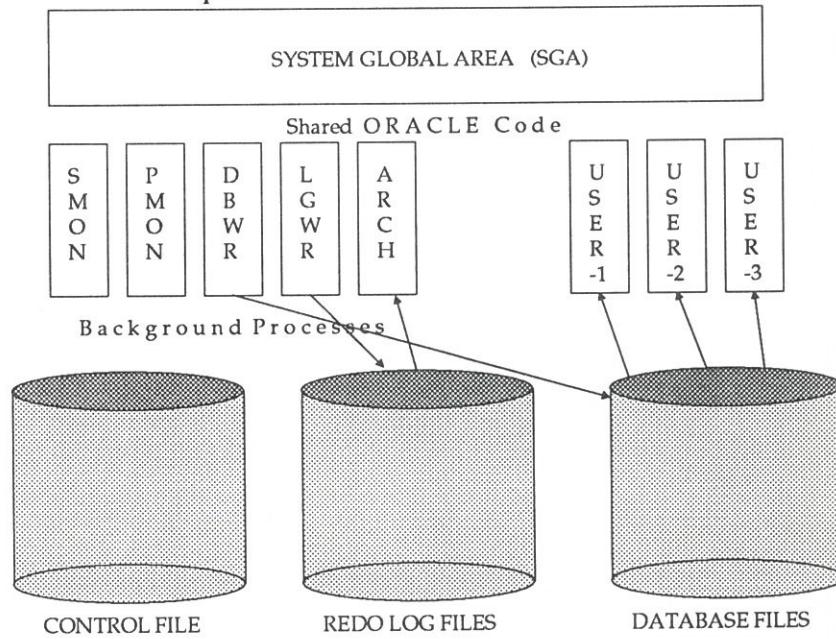
As Oracle Corporation introduces new products and enhances existing ones, the version numbers of the individual products increment independently. Thus, you might have a V5.1.12.2 RDBMS system working with SQL*Forms V2.0.3, SQL*Plus V1.0.9, and Pro*FORTRAN V1.1.2 (these numbers are used only for illustration).

What is an ORACLE Database System?

An ORACLE *database system* can be configured to provide a wide range of services. In all configurations, a user accesses a *database* (the database and redo log files) via an ORACLE *instance* (the software that manipulates the database).

Figure 1-1 illustrates a multiple user database system (one designed to be accessed by many users concurrently). Single user databases are somewhat simpler.

FIGURE 1-1
An ORACLE Database System



Instances and databases are independent of each other. An instance may be connected to exactly one database, although it can access other databases via SQL*Net. Establishing the connection is called *opening* the database. The database is *closed* when the instance no longer requires the connection.

When an instance is running and connected to a database, the instance is said to be *started*. When an instance is *shut down* (stopped), that particular database system is unavailable for use. At every startup, a database configuration file, typically called INIT.ORA, is read. This file contains several parameters used to start the database system.

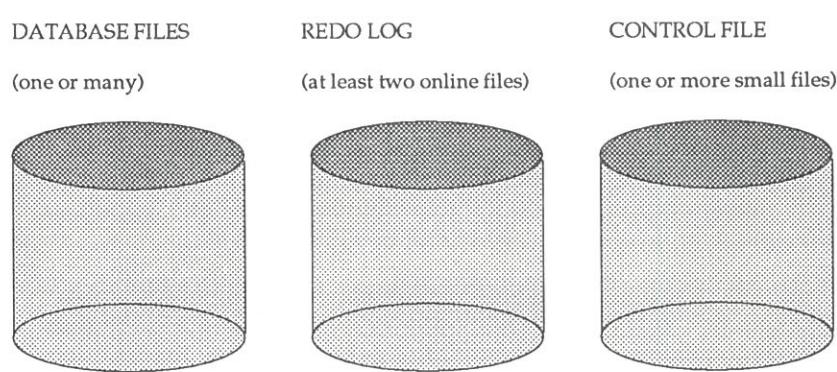
DBAs are responsible for starting and stopping instances. They do so using the DBA tool SQL*DBA. Frequently an instance is used with only one database, so the instance startup may automatically open the same database every time.

What is an ORACLE Database?

A *database* is a collection of data to be treated as a unit. A database is comprised of operating system files. Physically, there are *database files* and *redo log files*. Logically, the database files contain a set of dictionary and user tables, and the redo log files contain recovery data. Additionally, a database requires one or more copies of the *control file*. This file simply contains information that identifies and describes the rest of the database.

Figure 1-2 shows the requisite parts of an ORACLE database.

FIGURE 1-2
Components of an ORACLE Database



Before a database can be used to store data, it must go through a process called *database creation*. You need not necessarily create a database if you are moving from one version of ORACLE software to

another version. When database creation is necessary it is usually done as the last step of installing new ORACLE software. Chapter 13 describes the process of database creation and should be used in conjunction with the *Installation and User's Guides*.

A database is identified by a *database name* that it is given when it is created. The name is specified in the CREATE DATABASE statement and should be unique on a given CPU. Certain SQL and SQL*DBA statements used by the DBA will require the database name, but most users will not be concerned with the name.

A database can be described on many levels. The following chapters describe the physical and the logical structures which make up a database.

Chapter 3 describes the operating system files required by a database.

Chapter 4 describes the highest-level logical structures of a database: tablespaces and segments.

Chapter 5 describes logical structures at the user level (tables, clusters, indexes, and views).

Chapter 6 describes the ORACLE datatypes.

Chapter 7 describes the data dictionary and how it reflects the use of the database.

To be accessible, a database must be opened by an instance.

What is an ORACLE Instance?

An ORACLE *instance* provides the software mechanisms for accessing and controlling a database. An instance can be started independent of any database (that is, without mounting or opening any database). An instance can open at most one database (although in some operating systems, a single database can be opened by multiple instances).

An instance consists of:

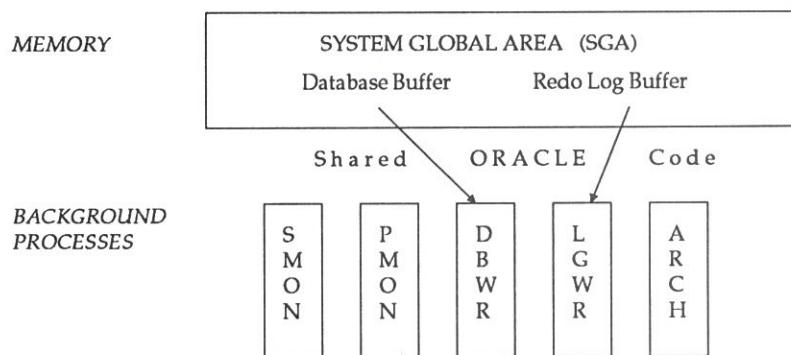
- a single shared memory area (called the *system global area* or the SGA) that provides communication between the processes. The SGA is described in more detail in Chapter 8.
- up to five background processes (referred to as DBWR, LGWR, SMON, PMON, and ARCH) that are shared by all users. The background processes are described in Chapter 9.

Chapter 14 discusses starting up and shutting down an ORACLE instance, and mounting, opening, closing, and dismounting databases.

Every instance needs access to the ORACLE RDBMS code. The program code for the RDBMS is referred to as the kernel. This code can be shared among multiple instances. Details about sharing can be found in the *Installation and User's Guides*.

Figure 1-3 shows the components of an ORACLE instance.

FIGURE 1-3
The Components of an
ORACLE Instance



Like databases, instances also have identifiers. All instances on one CPU must have distinct names. Instances are named when the ORACLE software is installed; how they are named is operating system dependent. Refer to the *Installation and User's Guide* for your operating system for more details about naming instances and setting the current instance.

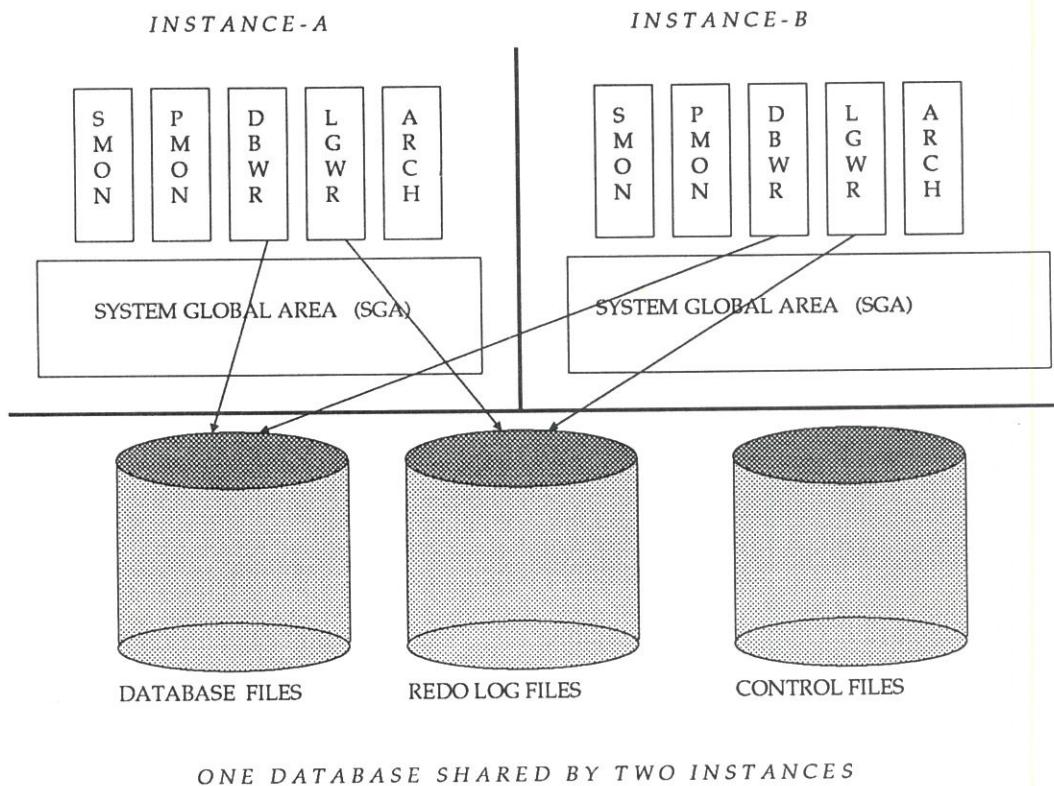
Shared Disk Database Systems

A database can be accessed by a single instance or, in some operating systems, by multiple instances simultaneously. If a database is accessed by more than one instance it is called a *shared disk system*. Chapter 21 addresses specific needs of shared disk systems.

Note that a shared disk system specifically addresses *one database* that is shared by multiple instances, whereas a distributed database requires *multiple databases and multiple instances*, that are connected by network software.

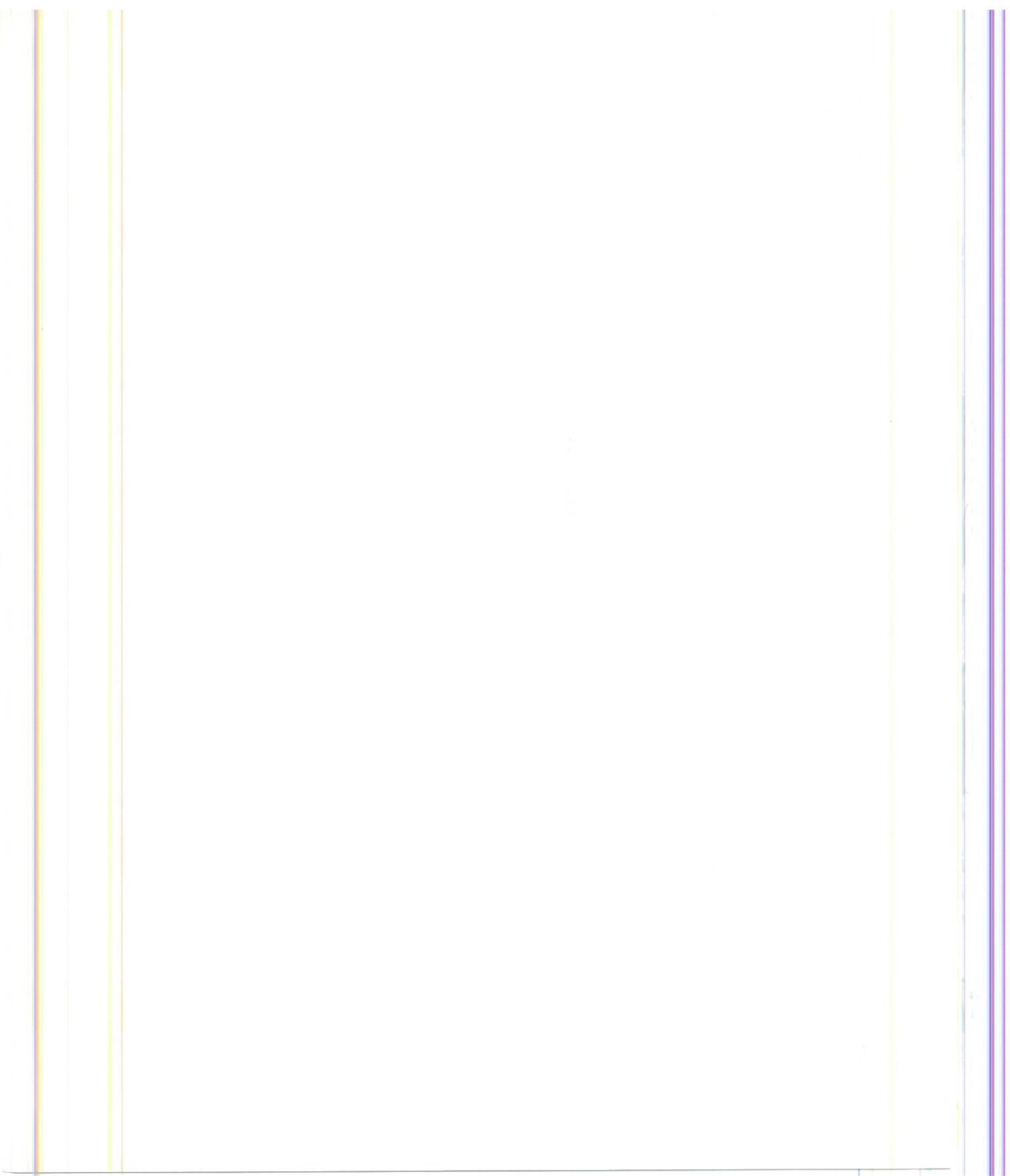
Figure 1-4 shows a shared disk system.

FIGURE 1-4
A Shared Disk
Database
System



Distributed Database Systems

An ORACLE database system can communicate with other databases via a network. Thus database users of one database system can access data from a different database. Chapter 22 contains information helpful for DBAs who are responsible for one or more databases connected by SQL*Net.



2

THE ROLE OF THE DATABASE ADMINISTRATOR (DBA)

*Thou dost preserve the stars from wrong;
And the most ancient heavens, through Thee, are fresh and strong.*
Wordsworth: *Ode to Duty*

*O Duty,
Why hast thou not the visage of a sweetie or a cutie?
Why glitter thy spectacles so ominously?
Why art thou clad so abominously?
Why art thou so different from Venus?
And why do thou and I have so few interests mutually in common
between us?*

Ogden Nash: *Kind of an Ode to Duty*

This chapter describes the role and responsibilities of the Database Administrator (DBA). In general, every database requires at least one person to perform DBA duties; if a database is large and supports many different applications, multiple DBAs may be more convenient.

This chapter describes:

- the general responsibilities of a DBA
- the two DBAs automatically enrolled in every database, having the usernames SYS and SYSTEM
- database privileges given only to DBAs
- the DBA utility SQL*DBA.

This chapter summarizes the DBA's responsibilities. The rest of this manual provides technical information and detailed instructions for performing DBA functions.

Who is the Database Administrator?

The (DBA) plays a very important role in the success of an RDBMS. As the primary "manager" of a database system, the DBA is responsible for seeing that the software and hardware forming the database system meet the needs of the users of the database. Thus, the DBA's concerns may include:

- software installation and maintenance
- database tuning for optimal database use
- database design
- data accuracy
- data completeness
- data security
- data storage
- data availability
- data recovery.

The DBA is frequently both the primary liaison with Oracle Corporation and the primary source of ORACLE information for users of the database.

The DBA need not be the person who installs ORACLE. However, it is frequently true that the DBA either controls or performs installations and software upgrades.

Ideally, the DBA has a good understanding of the database: for example, who the users are, what data they are storing and accessing and how often, what types of transactions are occurring, and how to maximize performance.

Strictly speaking, a DBA is any ORACLE user who has the *DBA privilege* on an ORACLE database (see Chapter 17). Each ORACLE system is installed with two DBA usernames, called SYS and SYSTEM, described in this chapter. Additional DBA usernames can always be added to an ORACLE database.

In this manual the term DBA usually implies a person responsible for the general DBA duties and who has an ORACLE username with DBA privileges.

Primary Responsibilities of the DBA

Some of the responsibilities of a DBA include:

- guaranteeing data integrity and consistency (see Chapters 11 and 12)
- monitoring and tuning performance (see Chapters 19 and 20)
- reducing unnecessary or redundant storage (see Chapter 16)
- facilitating sharing of data among users (see Chapter 12)
- guaranteeing database security (see Chapters 17 and 18)
- performing regular database backups (see Chapter 15)
- performing (software or media) recovery when necessary (see Chapter 15).

The DBA's Operating System Account

The DBA needs an operating system account or ID to gain access to the operating system under which ORACLE is running. The DBA's account may require more operating system privileges or access rights than normal database users require (for example, to perform certain steps of ORACLE software installation). Although the DBA need not own the files sent on the distribution media, he should have access to them.

In addition, the SQL*DBA program requires that the DBA's operating system account or ID be distinguished in some way in order to use the *system privileged* SQL*DBA commands. The method of distinguishing the DBA's account is operating system specific; for details please refer to the *Installation and User's Guide* for your operating system.

Access to the DBA's account (and thus to the privileged SQL*DBA commands) should be tightly controlled.

The Standard DBA Usernames

Every ORACLE system is automatically installed with two ORACLE users having DBA privileges: SYS and SYSTEM. They differ slightly in purpose, as described in the following sections.

If multiple persons will be performing DBA duties, it is preferable to create additional DBA usernames rather than have many users connect to a database through SYS or SYSTEM. This reduces the chance of inadvertently altering the tables or views owned by SYS or SYSTEM.

Some ORACLE utilities require tables or views owned by SYS or SYSTEM. For example, the CRT program and SQL*Forms product both use tables owned by SYSTEM. These tables are created when these products are installed; the tables should not be altered.

Username SYS

The ORACLE user SYS is enrolled with the password CHANGE_ON_INSTALL; the password should be changed immediately after database creation. Changing SYS's password is one of the last steps in the installation process, as documented in the *Installation and User's Guide* for each operating system.

The syntax is:

```
GRANT CONNECT TO SYS IDENTIFIED BY newpassword
```

Normal database users should *never* connect to the database with the SYS username and DBAs should do so rarely. SYS owns all of the base tables for the data dictionary and these tables are *critical* for RDBMS operation. SYS also owns the data dictionary views based upon those tables.

Information in SYS's tables is manipulated by the ORACLE RDBMS as a result of database use. Maintaining the integrity of the data dictionary is critical to ongoing database operation. Except as described in Chapter 7 under "Deleting Data Dictionary Items," **none of SYS's tables should be altered in any way**, nor should additional tables be created in SYS's account.

Note: Failure to follow these rules can result in the loss of all data in the database, requiring the DBA to recover the database.

By default only user SYS has access to the underlying tables used by the SQL*DBA MONITOR displays. See Appendix B for more information on MONITOR and granting other users access to these tables.

Username SYSTEM

The ORACLE user SYSTEM is enrolled with the password MANAGER; just as for SYS, the password should be changed immediately after database creation. Note that installation of some products like SQL*Forms may require or assume that SYSTEM's password is MANAGER.

Additional tables or views can be created by user SYSTEM. For example, a DBA connected as SYSTEM might reasonably create some additional views containing general database or user data. However, tables of interest only to individual users should not be created by SYSTEM.

Connecting as INTERNAL

In certain situations, the DBA will need to connect to the database using the keyword INTERNAL. Connecting as INTERNAL is equivalent to connecting as SYS, except no password is required. In this way, standard username validation and checking can be bypassed.

Connecting as INTERNAL is a *privileged* SQL*DBA operation. Thus, the ORACLE RDBMS will check that the operating system account attempting to connect as INTERNAL has the proper privileges to use the privileged SQL*DBA commands. See Appendix B for more information on the privileged SQL*DBA commands.

Connecting as INTERNAL is necessary when the database is mounted but not open (a relatively uncommon occurrence, but required for some database maintenance). Connecting as INTERNAL is required for the following operations:

- when creating a database (because the SQL statement CREATE DATABASE is used and the DBA must be "connected to a database", but the data dictionary has not yet been created to perform username validation)
- when altering a database to add or drop a redo log file using ALTER DATABASE.

Note: Connecting as INTERNAL at any other time is discouraged. Only DBAs should connect using INTERNAL.

DBA Privileges

The users SYS and SYSTEM have special privileges in the ORACLE system, as do any other users explicitly given DBA access to the database. For a description of these privileges, refer to "Users with DBA Privilege" in Chapter 17.

DBA Tools and the SQL*DBA Program

The most useful tool for a DBA is the SQL*DBA program. SQL*DBA may be used for many tasks, including:

- creating a new database
- starting and stopping an instance
- connecting as user INTERNAL when necessary
- changing the current host CPU in order to perform DBA functions on various database systems
- running SQL statements
- monitoring realtime use of the database
- archiving online redo log files
- recovering from instance or media failure.

Refer to Chapter 13 for instructions on using SQL*DBA to start up and shut down database systems, and to Appendix B for a description of SQL*DBA and complete syntax for the SQL*DBA commands.

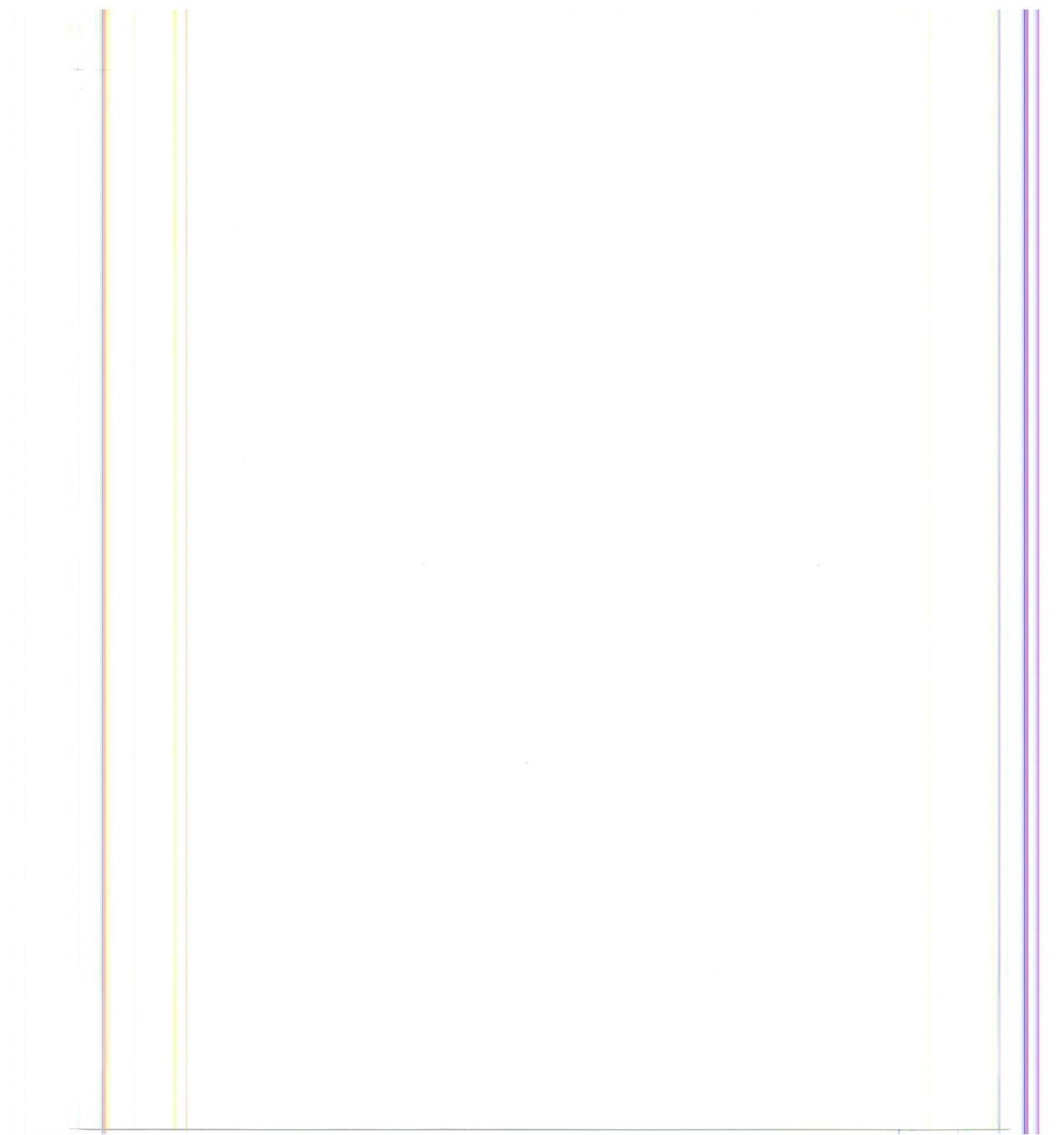
The data dictionary is also an important tool for DBAs. DBAs should be very comfortable interpreting and using the data dictionary tables and views. They should also feel free to "extend" the data dictionary, by creating new views or tables to provide information in formats they find useful. Like other users, DBAs are subject to some restrictions on altering the data dictionary; see Chapter 7.

The DBA is also likely to be concerned with the ORACLE utilities SQL*Loader, CRT, and Export/Import. These utilities are documented in the *ORACLE Utilities User's Guide*.

P A R T

II

ORACLE RDBMS CONCEPTS



3 ORACLE RDBMS FILE STRUCTURE

This chapter describes the operating system files required to run an ORACLE database system. Topics include the purpose and maintenance of the four types of files required by an ORACLE database:

- ORACLE RDBMS code files
- database files
- control files
- redo log files.

Typically only DBAs are concerned with physical files; database users rarely need to know about physical files. ORACLE RDBMS Version 6 allows the DBA much control over physical storage. For example, the DBA can partition data to spread I/O across devices or to enable part of a database to be offline or recovered while the rest of the database is online.

These features are especially important to DBAs managing large databases, as the DBA can manage portions of a database (for example, perform backup or problem resolution), while users can continue using the online part of the database.

Additional information relating to this chapter may be found in the following locations:

- Chapter 4 Tablespaces and Segments
- Chapter 15 Database Backup and Recovery
- Chapter 16 Space Management
- the *Installation and User's Guides* describe the specification, creation, and use of these files in detail specific to an operating system.

Figure 3-1 illustrates the minimum set of files required by an ORACLE database. As you can see, a database requires a minimum of four files in addition to the RDBMS software files. Many databases will add more files of each type.

FIGURE 3-1
Minimum Set of Files
Required by a Database System

Type of File	Minimum Number	Minimum Size
Database File	1	500 Kbyte
Online Redo Log	2	50 Kbyte
Control File	1	Size determined at installation (small)
ORACLE RDBMS program files	(many files)	Total space required is operating system dependent (estimate 17400 Kbytes)

The ORACLE RDBMS requires a fixed allocation of space to contain its data and programs; it does not simply allocate more disk space as it needs it. Instead, resources such as disk files are dedicated to the RDBMS to store database data, so the RDBMS knows it has the space exclusively.

Physical vs Logical Structures

Physical structures, such as operating system files, are stored on tangible hardware media (for example, magnetic tape, floppy disk, or a disk drive). A *file* corresponds to an allocation of space by the operating system. An ORACLE database system requires several files to run.

Normally, only the DBA is concerned with physical units of space (for example, making sure that the database files can hold all the database data, and adding another file when necessary). Occasionally a user might be concerned about physical structures, for example, for performance reasons such as minimizing I/O or disk contention.

Logical structures also correspond to units of space, but their boundaries are independent of physical space allocation. A logical *table*, for example, may span physical files. Examples of logical structures in ORACLE are tables and tablespaces.

Most database users are more concerned with logical database structures than with physical structures, because logical structures are more closely related to the data. For example, the speed with which data can be retrieved from a table may be a function of which tablespace the table resides in and whether indexes have been created on the table.

ORACLE RDBMS Program Files

You will receive Oracle Corporation software products on some type of release distribution media, typically magnetic tape or diskette. Your *Installation and User's Guide* provides instructions for installing ORACLE, including where to place the distribution files and how much space they require.

The release media contains many types of files, such as object libraries, executable files, and command files. The distributed set of ORACLE files is normally owned by either the DBA or a systems programmer or manager. Whether or not the DBA is responsible for performing installations and upgrades of ORACLE software, he may occasionally require files from the distribution media, for reference or for execution.

The release media should be kept in a safe and secure place in case future reloading is desired. If space is at a premium, the files can be backed up to tape after linking, as they are usually not required again.

Database Files

An ORACLE database consists of one or more *database files*. These files contain all the database *data*. The following are characteristics of database files:

- A file can be associated with one and only one database.
- One or more physical files form a logical unit of database storage called a *tablespace*.
- All database files for online tablespaces in a database must be accessible when an instance is started.
- Database performance is best if each database file is a logically contiguous allocation of space on disk; however, database files need not be contiguous.
- Once created, a database file must not change in size.

The first database file is the first file in the tablespace named SYSTEM. Multiple database files can be added to a tablespace, but a single file is associated with only one tablespace. For more information on how tablespaces use files see Chapter 4.

The steps that the DBA must take to prepare or identify database files vary by operating system; refer to the *Installation and User's Guide* for your operating system for details. On many operating systems, you need only name the files, and ORACLE automatically allocates and formats them. On some operating systems, the person installing the database must create database files before installation.

Number of Database Files

Only one database file is required; a small system might have just a single database file. Additional files can be added subject to two slightly different limits. In general, fewer larger files are preferred over smaller more numerous files.

The *absolute maximum* number of database files can be set using an optional argument to the CREATE DATABASE statement: MAXDATAFILES. The setting for MAXDATAFILES has some effect on the size of the control file. This maximum is subject to the ORACLE limit on the number of database files, which is 255, and to any operating system specific limits.

The *current maximum* number of database files is set by the INIT.ORA parameter DB_FILES. This limit is in effect every time an instance opens a database and remains until the database is closed. The default is 32. DB_FILES may temporarily reduce the limit set by MAXDATAFILES, but it cannot raise it.

The use of DB_FILES and MAXDATAFILES is optional. If neither are used, the default maximum number of database files is the ORACLE limit of 255 files.

Refer to Chapter 16 for instructions on adding new database files to new or existing tablespaces.

Size of Database Files

The first database file (the original SYSTEM tablespace) must be at least 500K bytes in order to contain:

- the initial data dictionary (see Chapter 7)
- the initial rollback segment (see Chapter 4).

The default database creation described in Chapter 13 creates a database file large enough for initial use of a database. Additional files of any size can easily be added subsequently (see Chapter 16).

If you install other ORACLE products, they may require space in the SYSTEM tablespace (online HELP for example); refer to the installation instructions for other products you plan to install.

Blocksize of the Database File

The ORACLE RDBMS manages the database files in units typically called *blocks* (sometimes called *pages*). *Database blocks* refer to blocks which correspond to the ORACLE *block size*; they may differ in size from the standard I/O block size of the host operating system.

For every operating system running the ORACLE RDBMS there is a default ORACLE block size. Typically, it is either 2K bytes or 4K bytes. Unless the majority of your records are very long or short, the default block size should not be altered. See the *Installation and User's Guide* for the default ORACLE block size for your operating system.

If you want to change the ORACLE block size, it is set at database creation using the optional INIT.ORA parameter DB_BLOCK_SIZE. The block size cannot be changed after database creation except by recreating the database. If you use a different block size for ORACLE than is used by your operating system it should be a multiple of your operating system's block size.

The parameter DB_BLOCK_SIZE also determines the *size* of the database buffers in the System Global Area (SGA). The parameter determining the *number* of those buffers, DB_BLOCK_BUFFERS, is the parameter having the most direct affect on system performance.

Online and Offline Database Files

Tablespaces can be taken offline (made unavailable to the RDBMS) or brought online at any time. Thus, all database files making up a tablespace are taken offline or brought online together. Single database files cannot be taken offline unless they are the sole file in a tablespace.

Any tablespace can be taken offline *with two exceptions*:

- The SYSTEM tablespace (and thus all database files constituting the SYSTEM tablespace) must always be online.
- Any tablespace containing a rollback segment that is active (in use by a running or suspended instance) must remain online.

To take tablespaces on and offline, use the ALTER TABLESPACE command (see Chapter 16).

Removing Database Files

All database files in a tablespace can be made temporarily unavailable, by taking a tablespace offline.

All database files in a tablespace can be removed from a database using the SQL statement DROP TABLESPACE (see Chapter 16). After successfully executing the statement, you can delete the corresponding database files using your operating system's file commands.

Control Files

Every time an ORACLE database is opened, one or more control files are checked. A *control file* is a small binary file, typically named something like ORACLE.DCF. A control file is associated with a single database only.

Control files are created during database creation. The current control file must always be accessible, as it is required when an instance starts in order to open and access a database.

Your choices regarding control files include:

- how many control files to maintain
- on which devices to place the files.

Control File Contents	<p>A control file contains information about the database that is required for the database to be accessed by an instance. For example, it contains the following types of information:</p> <ul style="list-style-type: none"> • names of physical database and redo log files • timestamp of database creation • database name. <p>The database name originates at database creation, from either the name specified by the INIT.ORA file parameter DB_NAME or the name used in the CREATE DATABASE statement.</p> <p>Control files are automatically modified by ORACLE; you cannot edit them. Because they are updated continuously during database use, they must be available for writing whenever the database is open.</p>
Number of Control Files	<p>Multiple identical copies of the control file are strongly recommended. By maintaining multiple control files on separate disks, you reduce the risk of media failure eradicating all control files. The only (slight) disadvantage of creating more control files is that certain operations (for example, checkpoints) require the update of control files, costing something in performance.</p> <p>The recommended configuration is to have a <i>minimum of two control files on different disks</i>.</p>
Names of Control Files	<p>The names of the control file(s) are indicated by the INIT.ORA parameter CONTROL_FILES. This parameter can indicate a list of one or more names for the control files; by default it lists one name. You can change the value (when the database is shut) to add more filenames or to change the names or locations of files.</p> <p>The instance startup procedure will recognize and maintain all listed control files but it will not create a new one. To add a new control file, you must copy a current control file to a new file and then add that file's name to the list of values for CONTROL_FILES, in between a database shutdown and startup.</p>
Size of Control Files	<p>Typical control files are very small. The main determinants of their size are the two INIT.ORA parameters LOG_FILES and DB_FILES.</p>

Redo Log Files

Every ORACLE database requires a redo log. The primary purpose of the *redo log* is to protect the database in the event of a system crash or media failure. The redo log records changes to be re-applied to the database if they were not written to the database for any reason (such as software or media failure). The process of re-applying changes is called *rolling forward*, or more generally, *recovery*.

The redo log consists of two or more operating system *redo log files* which are external to the database. At least two files are required to ensure that one is always available for writing while archiving the other. The point at which the system ends writing to one file and begins writing to another is called a *log switch*.

A running database system must always have write access to one online redo log file in order to record changes made by current users. If the ORACLE RDBMS cannot write to a redo log file for some reason, the database system will stop with an error.

Note: Redo log files are critical to database operation. The loss of a redo log file may necessitate reinstating the database to a prior state using a restore from a backup or a full database export.

The redo log can be used in two modes, ARCHIVELOG or NOARCHIVELOG mode, depending on whether the redo log data is archived. Note that all changes to the database are *logged* in the redo log files; the use of the online redo log files is not optional. A DBA may choose whether or not to *save* (archive) the log files, by using either ARCHIVELOG or NOARCHIVELOG mode.

How the Redo Log is Written

At any given time, only one redo log file is being written (no matter how many instances are accessing the database). The current online redo log file is written by the background process LGWR (see Chapter 9). Entries are buffered into the SGA and then written to the online redo log file sequentially in blocks, by LGWR.

Regardless of the mode chosen, online log redo files are written in a "circular" fashion. If ARCHIVELOG mode is enabled, the online redo log files are a temporary storage area for movement of data to the offline log; no log file can be re-written until its contents are archived. If NOARCHIVELOG mode is enabled, when the last online redo log file fills, writing continues by returning to overwrite the contents of the first file.

Redo Log File Contents The primary contents of redo log files are the *redo entries*, which record roll forward data to reconstruct all changes made to the database. Because rollback data (see Chapter 4) is stored in the database in rollback segments, the redo log also protects rollback data. In addition to these entries, a redo log file contains administrative information, such as the state of a database file vis a vis the instance writing it and the last checkpoint taken.

Because redo log files store low-level representations of changes that cannot be related to user actions, the contents of a redo log file should never be edited, altered, nor used for any application purposes, such as auditing.

Each file is identified by a *log sequence number*, which is used during recovery.

Choosing the Log's Mode of Use

The redo log can be used in two modes. The mode is chosen when CREATE DATABASE is invoked, and can be switched any subsequent time the database is closed with the SQL statement ALTER DATABASE.

NOARCHIVELOG This mode prepares only for instance recovery, not for recovery from media failure, as only the most recent changes to the database are available at any given time. Redo log files are not saved, once filled.

ARCHIVELOG This mode provides protection against media failure as well as instance failure, because all changes to the database are saved. Each redo log file must be *archived* (usually to tape) before it can be reused.

All changes made to the database are logged, regardless of the mode chosen. Information necessary for instance recovery (recovery from either system crash or software failure) can always be found in the redo log, regardless of the mode chosen.

However, if you are willing to assume the extra responsibilities associated with archiving redo log data (such as tape writing and storage, regular backups, and record keeping), you gain the added security of knowing that you can reconstruct a portion of your database contained in damaged disk files without losing any transactions and with a minimum of effort.

Using the Redo Log in NOARCHIVELOG Mode

NOARCHIVELOG mode is the minimum method of protection to guarantee database integrity. It protects against system and software crashes but does not protect against media failure. In NOARCHIVELOG mode, the redo log files are written exactly as in ARCHIVELOG mode, but an individual redo log file can be reused.

without saving its contents offline, so there is no offline log. Thus, only the most recent changes are available at any time. This has two major implications:

1. The redo log can be used only to recover from failures that occur while the necessary log is still online. Instance failure can always be recovered from using the online log. However, media failure cannot be recovered from, because not all of the required roll-forward data has been saved.
2. If a tablespace becomes unavailable due to some failure, the entire database cannot be accessed until the tablespace has been recovered. Otherwise, if database operations were allowed to continue, information required for tablespace recovery might be overwritten in the online redo log file.

Using the Redo Log in ARCHIVELOG Mode

In ARCHIVELOG mode, all changes to the database are saved (archived) so that they can be reapplied in the event of media failure. As with NOARCHIVELOG mode, the redo log also can be used for instance recovery after a system or software crash. Archiving requires the use of an online and an offline redo log:

online redo log	the set of current redo log files containing the newest log data and available to the RDBMS for writing.
offline redo log	the set of all older redo log files which have been archived, usually offline on tape.

In ARCHIVELOG mode, when an individual online log file fills, the redo entries it contains must be archived before the log file can be reused. (In NOARCHIVELOG mode, a file need not be archived before it is reused.)

The use of ARCHIVELOG mode in preparation for media recovery also requires periodic physical backups of the database. Backups should be taken regularly, using an operating system backup utility. You may backup the entire database a file at a time, even while the database is open and users are making updates. The redo log records all transactions following that snapshot, until the next snapshot is taken. Adequate offline storage is thus required to store all changes which occur in each interval between backups.

For additional information about using ARCHIVELOG refer to Chapter 15.

Configuring Redo Log Files When configuring redo log files you may consider:

- How much total redo log space do you need?
- How many online redo log files do you want?
- Are you using ARCHIVELOG mode?
- If so, would you rather archive large files less often, or smaller files more often?
- How many redo log files can be archived per archive tape?
- On which devices should you place the files?

These issues are discussed in detail in Chapter 15.

Size of Redo Log Files

Redo log files need not all be the same size. The minimum size for a redo log file is 50K bytes. The size of the redo log file is set in operating system blocks, not ORACLE block sizes. The block size is significant primarily if you decide to change the value used for the INIT.ORA parameter LOG_CHECKPOINT_INTERVAL because the value is the number of operating system blocks, not ORACLE blocks.

Number of Redo Log Files

Two redo log files are required by every database. Additional files can be added subject to two slightly different limits:

The *absolute maximum* number of redo log files can be set using an optional argument to the CREATE DATABASE statement: MAXLOGFILES. The setting for MAXLOGFILES has some effect on the size of the control file. This maximum is subject to the ORACLE limit on the number of log files, which is 255.

The *current maximum* number of log files is set by the INIT.ORA parameter LOG_FILES. This limit is in effect every time an instance opens a database and until the database is closed. The default is 16. LOG_FILES may temporarily reduce the limit set by MAXLOGFILES, but it cannot raise it.

The use of LOG_FILES and MAXLOGFILES is optional. If neither are used, the default limit to the number of redo log files is the ORACLE limit of 255 files.

Checkpoints

When an online redo log file fills, it triggers an action called a *checkpoint*. Checkpoints occur in both ARCHIVELOG mode and NOARCHIVELOG mode. By the time a checkpoint has completed, all the modified database buffers will have been written to the database files assuring that all changes have been written to disk. Checkpoints have no effect on current database users or their transactions, but guarantee that earlier log entries are no longer needed for instance recovery and that all changed database blocks are written to disk.

Checkpoints occur when log files fill, and may also occur at specified intervals during the writing of redo log files. By periodically saving the changes to the database, checkpoints serve two purposes:

- decrease the time required by instance recovery if it should be required
- mark an online redo log file as ready for archiving or for rewriting.

For more information on checkpoints, refer to Chapter 15.

4

TABLESPACES AND SEGMENTS

'Contrariwise,' continued Tweedledee, 'if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic.'

Lewis Carroll: *Through the Looking-Glass IV*

This chapter contains topics of interest to advanced ORACLE users and application designers as well as to DBAs. Topics focus on the logical structures which constitute an ORACLE database and include:

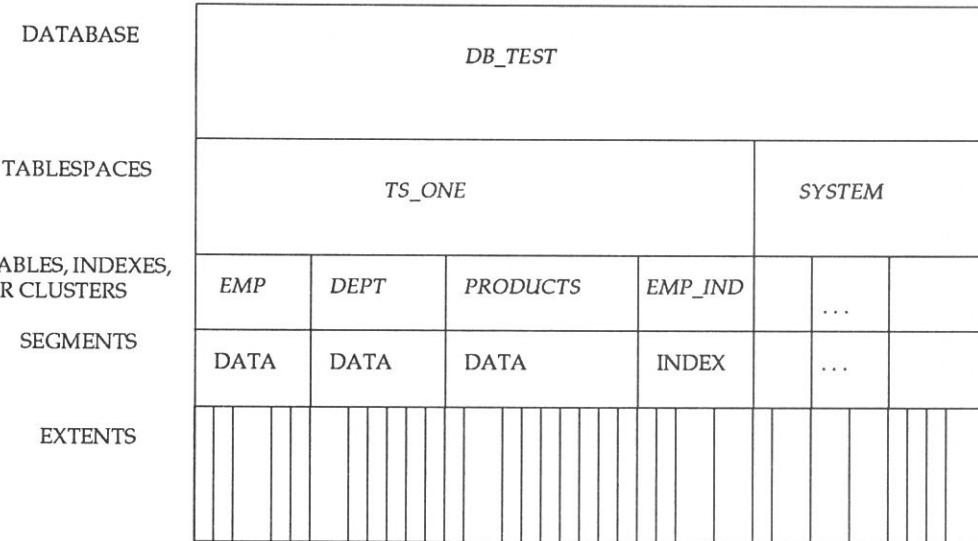
- tablespaces
- the five types of segments:
 - data segments
 - index segments
 - rollback segment
 - temporary segments
 - bootstrap segment
- storage parameters used to define how segments allocate units of space called extents.

You may find related information in the following places:

- Chapter 5 ORACLE Table Structure
- Chapter 14 Database Startup and Shutdown
- Chapter 16 Space Management.

Figure 4-1 illustrates the relationship of the logical structures discussed in this chapter.

FIGURE 4-1
Logical Structures
Forming a Database



Each database is identified by a database name, specified in the CREATE DATABASE statement.

The discussion of logical database structures continues in Chapter 5, which discusses tables, clusters, views, and indexes.

Tablespaces

A database is divided into logical divisions called *tablespaces*. A database may have one or more tablespaces; the original tablespace is named SYSTEM. Each logical tablespace corresponds to one or more physical database files.

As a basic ORACLE logical unit, the tablespace is of primary concern to DBAs. DBAs can use tablespaces to:

- control space allocation for database objects, such as tables, indexes, and temporary segments
- set space quotas for database users
- control availability of data, by taking individual tablespaces online or offline
- backup or recover data
- allocate data across devices, to improve performance.

Each tablespace has a set of default storage characteristics (described later in this chapter), that apply to tables or indexes created in that tablespace. When you create a table or index you can override the defaults if you want the object to use different storage parameters.

The Relationship between Database Files and Tablespaces

Each logical tablespace corresponds to one or more physical database files. For example, the simplest database would have one tablespace, with one database file. To enlarge that database, a second database file could be added to that tablespace, or an entirely new tablespace (and file) could be added. Thus, one or more database files provide the actual storage for a tablespace.

SQL statements are used to add tablespaces and files (see ADD TABLESPACE or ALTER TABLESPACE).

The SYSTEM Tablespace

The SYSTEM tablespace exists in every database. It is automatically created by CREATE DATABASE. A small database might have only one tablespace, named SYSTEM. The SYSTEM tablespace always contains the data dictionary for the entire database, with names and locations of all database objects (such as user tables, indexes, and other tablespaces).

The SYSTEM tablespace must always be online; if you try to take it offline an error is returned.

Like other tablespaces, the SYSTEM tablespace can be enlarged by adding new database files to it.

If the SYSTEM tablespace is to be the only tablespace, it must be large enough to contain:

- the data dictionary
- all rollback, bootstrap, and temporary segments
- online HELP if loaded
- tables required by other ORACLE products installed
- all user-created objects.

Using Multiple Tablespaces

Most DBAs find that using multiple tablespaces (in addition to the SYSTEM tablespace) allows more flexibility regarding database use. For example, when you have multiple tablespaces, you can:

- spread I/O across tablespaces (database files) based on application use
- separate table data from index data, reducing contention
- enforce space allocation schemes for users or applications
- take individual tablespaces offline while others are online
- reserve different tablespaces for different types of database use (such as high update activity, read-only activity, or temporary work areas).

One example of such added flexibility is a database with two or three tablespaces, with one containing only additional rollback segments. This configuration allows a tablespace to be brought offline while the rest of the database is online. If only one tablespace (SYSTEM) is used, then either the whole database must be accessible, or none of it is; no partial availability is possible, as the SYSTEM tablespace must always be available.

Associating Users with Tablespaces

ORACLE users who will be creating database objects must have the right to create objects in a given tablespace. For each user, a DBA can:

- grant RESOURCE privilege (the ability to create and store objects) on one or more tablespaces
- assign a default tablespace (for tables and other objects)
- assign a quota for each tablespace (how much space a user can use for table and index data in that tablespace)
- assign a default temporary tablespace (for any temporary segments created by ORACLE on behalf of users' queries).

See Chapter 17 for descriptions of the SQL statements GRANT and ALTER USER.

Directing Tables to Tablespaces

If you have RESOURCE privilege on multiple tablespaces, then when you create a table or an index, you can specify which tablespace should contain the table data or index data.

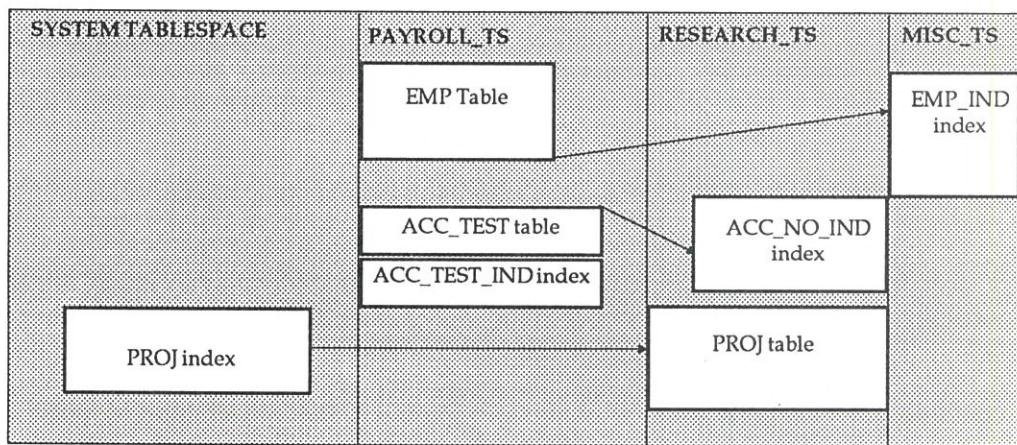
By choosing which tablespaces should store given tables or indexes, you can:

- choose which tables are stored together in individual database files (if only one file is added to any tablespace)
- reduce disk drive contention for the data and indexes of one table by storing the table data and the index data in different tablespaces
- control which physical storage devices (and therefore, the associated speed of access) will store which data
- use default storage parameters for data and index segments
- group together tables with high activity so they can be backed up more often
- group together all tables belonging to one application to coordinate access (to make sure they are always accessible and backed up together)
- reduce head contention for several tables.

A tablespace can contain many tables and indexes. A table or index cannot span tablespaces, but it may span the files making up a tablespace.

Figure 4-2 shows the placement of several tables and indexes across tablespaces.

FIGURE 4-2
Using Tablespaces
to Store Table and
Index Data



See Chapter 16 for more hints on choosing tablespaces when creating tables or indexes.

Online and Offline Tablespaces

Using the SQL statement `ALTER TABLESPACE`, a DBA can bring any tablespace online or offline whenever the database is open. The only exception is that the **SYSTEM tablespace must always be online**.

You may wish to take tablespaces offline for any of the following reasons:

- in general, to make a portion of the database unavailable, but allow normal access to the remainder of the database
- to perform a tablespace backup (although a tablespace can be backed up while online and in use)
- to make an application (and its related group of tables) temporarily unavailable while updating or maintaining the application
- to make the best use of currently available storage devices.

The ability to take tablespaces online and offline allows you more options in dealing with infrequently used data or a temporary shortage of storage devices.

The data dictionary in the SYSTEM tablespace records whether a tablespace is offline or online. If a tablespace was offline when a database was dismounted, it will be offline when the database is subsequently mounted.

Offline Tablespaces

When a tablespace is offline, the files making up that tablespace are not open, and data in that tablespace cannot be accessed.

A tablespace can only be brought online by the database which took it offline, as the necessary data dictionary information is maintained in the SYSTEM tablespace of that database. An offline tablespace cannot be read or edited by any utility other than ORACLE. Thus, tablespaces cannot be transferred from database to database; to transfer data you may use Export and Import.

A tablespace may be automatically changed from online to offline when certain errors are encountered (if, for example, the database writer process, DBWR, fails several attempts to write to the tablespace). Users trying to access tables in the tablespace with the problem will receive an error. Normally, a restart of the instance will clear the problem with the tablespace.

"Accessing" Offline Data

When a table's data and indexes are distributed across tablespaces, some of which are offline, you may wonder if you can access the data that is currently online. Though any SQL statement requiring offline data will fail, some SQL statements you might expect to fail may work,

depending on what data is online or what data is accessible through the SGA.

Generally speaking, if a tablespace is offline and a table has one index stored in that tablespace, then many statements referring to that table probably cannot be executed while the tablespace is offline. For example, an update requiring a change to index values would fail.

However, some SQL statements may succeed, if the optimizer determines that the offline index is not required. For example, if the optimizer determines that a full table scan rather than an index will be used for retrieval, then a query could successfully execute. You can often reword the query to suppress the use of an index.

Or, if a table is offline, but an index is online that indexes all the columns referenced in a query, then that query may also complete successfully.

In other words, if the optimizer determines that it has access to enough data to execute a statement, it will do so. If data in the offline tablespace is required, then the statement will fail.

Recovering Offline Tablespaces

While offline, a tablespace may have recovery operations performed on it. Thus, a database can remain operational even with a damaged file, because other files (in other tablespaces) can be accessed while the tablespace containing the damaged file is taken offline and recovered.

For example, if a disk error occurs during normal operation, you can bring the affected tablespace offline, causing a rollback of all transactions currently active in that tablespace. You can then restore a backup copy of the tablespace (perhaps to another disk) and use SQL*DBA to recover the tablespace. Finally, you would bring the tablespace back online.

Separate recovery is also useful during instance recovery, as individual parts of the database can be made available before the entire database is recovered. That is, an instance can be started, and the database mounted and opened, as soon as the SYSTEM tablespace is recovered but before other tablespaces are recovered.

Adding New Files or Tablespaces

You may add database files to enlarge existing tablespaces, or create new tablespaces in order to enlarge the database or to allocate space to certain applications. To add another database file to an existing tablespace, use the SQL statement ALTER TABLESPACE ADD FILE.

To create a new tablespace, use the SQL statement CREATE TABLESPACE. Refer to Chapter 16 for instructions on both operations.

Segments and Extents

All data in a tablespace is stored in allocations of database space called segments. A *segment* is a set of database blocks allocated for storage of database data. That data may be table or index data or it may be temporary data required by the RDBMS to operate. Segments are the next logical level of storage after tablespaces. A segment cannot span more than one tablespace, but it can span files within a tablespace.

A database requires up to five types of segments:

- data segments
- index segments
- rollback segments
- temporary segments
- bootstrap segment.

Segments are created in different ways depending on the type of segment. Most users are concerned with data and index segments only; these segments are created when users execute CREATE TABLE and CREATE INDEX statements.

Generally, only DBAs are concerned with rollback, temporary, and bootstrap segments. DBAs control rollback segments directly, using SQL statements; the RDBMS automatically creates temporary segments and the bootstrap segment.

All segments are structured similarly and are made up of extents. Most segments begin at some specified size (number of extents) and grow dynamically (adding extents), as required. The next section describes the storage parameters common to all segments.

Following the discussion of storage are detailed descriptions of each of the five types of segments.

Segment Storage Parameters

Every segment is defined by several *storage parameters*. These parameters are associated with each tablespace (via CREATE TABLESPACE) and determine the default space allocation for all segments created in that tablespace. Although the storage parameters apply to all types of segments, the following examples use data and index segments for description purposes.

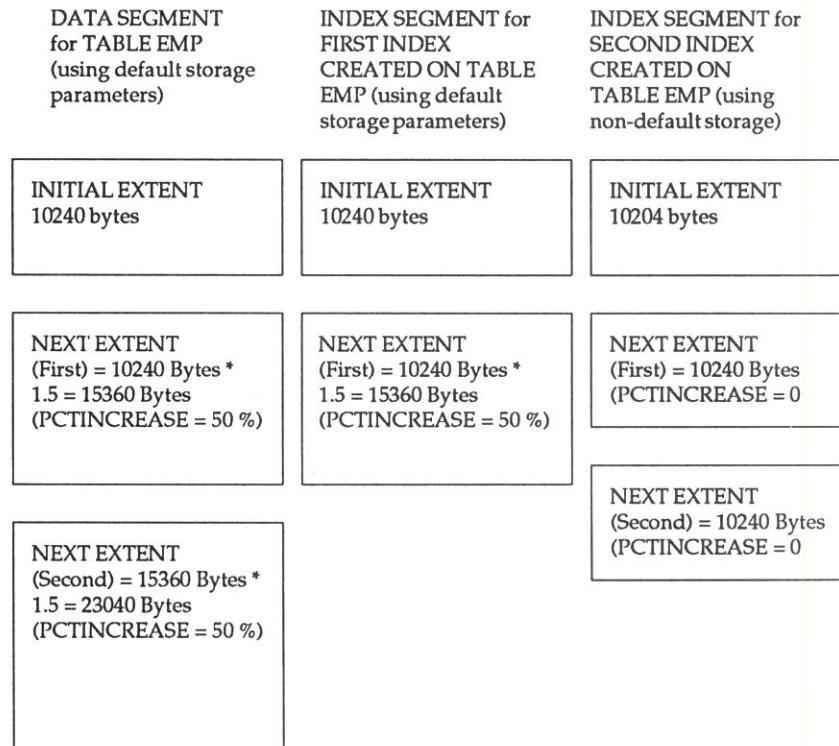
The purpose of storage parameters is to control how a table (or index) will obtain database space for its data. For example, you can set how much space is initially reserved for the table's data or limit the maximum amount of space a table will use.

Storage parameters are expressed in terms of extents of data. An *extent* is an allocation of contiguous database space, expressed as a number of bytes of data.

For example, when a table is created, its data segment contains an *initial extent* of a specified number of bytes. Although no rows have been inserted yet, those ORACLE blocks are reserved for that table's rows.

As rows are inserted, the table will grow and may eventually fill the initial extent. When more space is required for new data, a *subsequent extent* of blocks is allocated, so now the data segment has two extents. Figure 4-3 shows a set of extents for table EMP.

FIGURE 4-3
Extents for Table EMP (Using Default Parameters)



Storage parameters apply to all types of segments or objects created in a tablespace (primarily tables and indexes). For INITIAL and NEXT, the abbreviations K or M may be used for kilobytes or megabytes and values are rounded up to multiples of DB_BLOCK_SIZE. Descriptions of the storage parameters follow.

INITIAL	The size, in bytes, of the first extent allocated when the table is created — or, the original amount of space available for table data. The default initial extent size is 10240 bytes.
NEXT	The size, in bytes, of the next extent to be allocated. This size is a base value, which may be increased by the percentage given for PCTINCREASE. The default is 10240 bytes.
MAXEXTENTS	The total number of extents, including the first, which can ever be allocated. The default is 99 extents; the absolute maximum varies by operating system.
MINEXTENTS	<p>The total number of extents to be allocated when the segment is created. This allows for a large allocation of space at create time, even if the space is not contiguous. The default is 1 extent (just the initial extent is allocated).</p> <p>If MINEXTENTS is greater than 1, then the specified number of subsequent extents are allocated at CREATE time using the values INITIAL, NEXT, and PCTINCREASE.</p> <p>You might want to increase the value for MINEXTENTS when you are creating a rollback segment or when your database is "fragmented" and you want to ensure that you will have enough space to load all the data for one table.</p>
PCTINCREASE	<p>The percent by which each NEXT extent will grow over the last extent allocated. If PCTINCREASE is zero, then the size of each additional extent remains constant. However, if PCTINCREASE is positive, then each time NEXT is calculated, it will grow by PCTINCREASE. PCTINCREASE cannot be less than zero.</p> <p>The result of PCTINCREASE multiplied by the size of the last extent is rounded up to the next multiple of a block size. The default is 50.</p> <p>Note: Do not confuse PCTINCREASE with PCTFREE and PCTUSED, which are described in Chapter 5.</p>

Which Storage Parameters are in Effect?

The storage parameters actually in effect for a given segment may be determined at several levels. Storage parameters can be set in the following ways, from the most specific to the most general way:

- in an ALTER statement (for any of the objects TABLE , INDEX, CLUSTER or ROLLBACK SEGMENT)
- in the CREATE statement (for the same objects)
- in the ALTER TABLESPACE statement
- in the CREATE TABLESPACE statement.

That is, any storage parameter specified at the object level will override the corresponding parameter set at the tablespace level. If parameters are altered, only the extents not yet allocated are affected. When storage parameters are not explicitly set at either level, ORACLE system defaults apply.

For examples of specifying and altering these parameters via the CREATE TABLE and ALTER TABLE commands, refer to Chapter 16.

How are Extents Allocated?

Each segment contains a *segment header block* that describes the characteristics of that segment and contains a directory of the extents in the segment. Similar information is contained in the data dictionary view DBA_EXTENTS, which lists all extents in a database, no matter what tablespace they reside in. (The data dictionary indicates free space and is used to find new extents.)

When the ORACLE RDBMS needs to allocate another extent for a segment, it searches through the "free space" in the tablespace and allocates the first contiguous set of ORACLE blocks of the required size that it finds. Then the segment header and data dictionary are updated to show that a new extent has been allocated and that space is no longer free.

When are Extents Deallocated?

Data and index extents are not deallocated until the data or index segment is deallocated. As long as a table exists, any data block allocated to it remains allocated for that table; new rows for that table can be inserted to a block if there is enough room. Even if all rows of a table are deleted, the data blocks are not reclaimed for use by other objects in the tablespace.

When a table is dropped, all its data and index segments are reclaimed by the tablespaces they were in, and the extents are made available for other objects in the tablespaces.

Likewise, in an index segment, all blocks allocated to an extent remain allocated as long as the index exists. When the index is dropped, the blocks are reclaimed for other uses within the tablespace.

When extents are freed because the segment is dropped, the data dictionary is updated to reflect the regained extents as available space.

Data and Index Segments

Of most interest to users are segments that contain table and index data:

data segments A data segment contains all the data for one table (or cluster). A table (or cluster) always has one single data segment.

index segments An index segment contains all the index data for **one** index created for a table or cluster. One table may have one, many, or no index segments associated with it, depending on how many indexes it has. A cluster must have at least one index segment — for the cluster index created on the cluster key.

To see how table EMP and its related indexes might be stored in various segments and tablespaces, refer to Figure 4-3.

If you have RESOURCE access to multiple tablespaces, you can create an index in a different tablespace than the one where the table was created. You have similar control when creating cluster indexes.

Rollback Segments

Each database contains one or more rollback segments. A *rollback segment* is a portion of the database which records actions which should be undone under certain circumstances. Rollback segments are used to provide:

- transaction rollback
- read consistency
- recovery.

A rollback segment contains rollback entries for one instance's transactions, which may affect database objects in any tablespace in the database.

An instance cannot start unless it has access to at least one rollback segment. When multiple instances share a database, each requires its

own set of rollback segments. Rollback segments (with the exception of the SYSTEM rollback segment) cannot be shared by instances.

The DBA uses SQL statements to create, drop, or increase the size of rollback segments:

```
CREATE ROLLBACK SEGMENT ...  
ALTER ROLLBACK SEGMENT ...  
DROP ROLLBACK SEGMENT ...
```

Refer to Chapter 16 for a description of these operations.

The Rollback Segment SYSTEM

An initial rollback segment, called SYSTEM, is created when a database is created. The initial segment is created in the SYSTEM tablespace using the default storage parameters associated with that tablespace. This size is sufficient for small databases, but many DBAs will want to enlarge the original rollback segment, or add new ones to accommodate the expected number of users and transactions.

An instance always acquires the SYSTEM rollback segment in addition to any rollback segments it needs. However, the RDBMS uses the SYSTEM rollback segment only for some special system transactions and distributes user transactions among the non-SYSTEM rollback segments only.

Multiple Rollback Segments

Multiple rollback segments are usually preferred to one rollback segment to improve performance. Multiple rollback segments are required when:

- a database has multiple tablespaces
- multiple instances are accessing a database running in shared disk mode
- the database will be accessed by many simultaneous users.

If a database has multiple tablespaces, it must have two or more rollback segments. Each tablespace does not require its own rollback segment. For example, if a database has three tablespaces, then the SYSTEM tablespace might include both the original rollback segment named SYSTEM and a public rollback segment named R_SEG1 while either one of the other tablespaces could contain R_SEG2.

If a database is running in shared disk mode, each instance requires access to its own rollback segment in order to start. For additional details, see Chapters 15 and 21.

Contents of the Rollback Segment

The rollback segment consists of several rollback entries. Among other information, a *rollback entry* includes a transaction ID, block information, and a description of the data as it existed before the transaction's operation. The description is actually the inverse operation of the transaction. Rollback entries for an individual transaction are linked together by transaction ID so the entries can easily be found if necessary for rollback.

Rollback segments cannot be accessed or read by database users or the DBA; they are written and read by the RDBMS only.

Rollback entries are actually recorded twice. First they are written to the rollback segment. Then, as they are database entries, these writes are also recorded in the online redo log file. Thus, data in the rollback segments is protected by the redo log. This assures that rollbacks for active transactions can always be performed, even after a system crash where the rollback segment blocks were not yet written to disk.

Like other segments, the rollback segment is allocated in groups of blocks called *extents*. In contrast to data and index extents, rollback segment extents contain relatively temporary information. This allows rollback segments to be treated as a "ring of extents," allowing the reuse of blocks in extents in a circular fashion.

A rollback segment extent is only freed and available for other use in the tablespace when the rollback segment is dropped.

How Segments are Allocated to Active Transactions

Each rollback segment can handle a certain number of transactions from one instance. That number is approximate and is a function of the ORACLE block size. The RDBMS distributes active transactions across available segments, so the more rollback segments that exist, the less contention there will be for any one segment.

For the following discussion, assume that we have three transactions, T1, T2, and T3, and one rollback segment with two currently allocated extents E1 and E2.

All transactions write sequentially to the same segment. Each transaction writes to only one extent at any given time. Thus, assume that transactions T1, T2, and T3 are all currently writing to extent E2.

When a transaction runs out of space in the current extent and needs to write to a different extent, the RDBMS must find an available extent in which to write. The RDBMS may either reuse an extent already allocated to the rollback segment or acquire a new extent.

If there are no *active rollback entries* in the next rollback segment extent already allocated in the ring, the RDBMS will use that extent.

Information in an extent is active if it reflects transactions which are still active and have not been committed or rolled back.

The RDBMS will always try to reuse the next extent in the ring first. However, if that extent still has active data, then the RDBMS must allocate a new extent, until the number of extents reaches MAXEXTENTS.

Thus, if T1 needs more space for rollback entries, it may reuse extent E1 only if there are no active entries in E1, or it may allocate extent E3.

When Rollback Information is Required

Each rollback segment maintains a table of all currently active transactions in the SGA as well as rollback entries for each change performed in those transactions.

Rollback segments record the data prior to change, on a per transaction basis. For every transaction, each new change is linked to the previous change. If the transaction needs to be rolled back, the changes in the chain are applied to the database blocks in an order that restores the data to its previous state.

Similarly, when ORACLE needs to provide a read consistent snapshot, it uses the rollback segments to create older versions of data and index blocks. (See Chapter 12 for a discussion of read consistency.)

All types of rollbacks use the same procedures:

- statement level rollback (due to deadlock or statement execution error)
- rollback to a savepoint
- rollback of a transaction due to user request
- rollback of a transaction due to abnormal process termination
- rollback of all outstanding transactions when an instance terminates abnormally.

See Chapter 11 for a discussion of rolling back work.

Size of Rollback Segments

If a system is running only short transactions, rollback segments should be small so they are always cached in main memory. The main disadvantage of small rollback segments is the increased likelihood of the error "snapshot is too old" when running a long query involving records that are frequently updated by other transactions. This error occurs because the rollback entries needed for read consistency are overwritten as other update entries wrap around the rollback segment.

For one instance rollback segments should be about the same size; it is not beneficial to have some large and some small segments, as the RDBMS distributes transactions evenly across segments regardless of the segment size.

Generally speaking, you should not set a low MAXEXTENTS for rollback segments.

Private and Public Rollback Segments

A rollback segment can be either **private** or **public**. Any number of either type of segment may exist in a database, as long as every instance that opens the database can access at least one rollback segment.

If your database will never be mounted SHARED (in general, this is true for all non-VAX systems) you should create all rollback segments as PUBLIC and ignore the following discussion about PRIVATE rollback segments.

In shared disk systems, it may be useful to use private rollback segments. See the following sections and Chapter 21 for more information on private and public rollback segments.

For a rollback segment to be *used*, first a DBA must create it, and then an instance must startup in order to claim it. An already running instance cannot use newly created rollback segments without being shut down and restarted.

At startup, after determining the number of rollback segments required, an instance tries to acquire that number of segments. First it acquires all private segments named by the INIT.ORA parameter ROLLBACK_SEGMENTS, and then, if necessary, it acquires some number of public rollback segments that are not currently in use by other instances.

Public Rollback Segments

A **public rollback segment** is one which is generally available for use by any instance accessing a database. A DBA can create a public rollback segment using the optional keyword PUBLIC in the CREATE ROLLBACK SEGMENT statement.

If an instance requires a rollback segment and has already used its private segment(s), it will look for a public segment. Once a public rollback segment has been "claimed" by an instance, it cannot be used by any other instance until that instance shuts down and closes the database. At that point the public rollback segment is again available for use by the next instance to claim it.

A shared database could have only public segments, as long as the number of segments is high enough to assure that each instance would be able to startup with the right number of rollback segments.

Private Rollback Segments	Private rollback segments may be useful to satisfy local needs, such as rollback segments that must be particularly small or large or on local disks. A <i>private rollback segment</i> is created without the keyword PUBLIC and is used only if it has been named in the INIT.ORA file for the parameter ROLLBACK_SEGMENTS.
	If a rollback segment is created without the keyword PUBLIC but no running instance has named it in the INIT.ORA file, then that rollback segment is dormant.
	If one instance attempts to acquire a rollback segment by specifying its name in the INIT.ORA file, but a second running instance has already acquired the instance, the first instance will receive an error on startup.

Deferred Rollback Segments	When a tablespace goes offline such that transactions cannot be rolled back immediately, a <i>deferred rollback segment</i> is written. It contains the rollback entries that could not be applied to the tablespace, so they can be applied when the tablespace comes back online. These segments disappear when the tablespaces are recovered.
	Deferred rollback segments are automatically created in the SYSTEM tablespace. The format of the deferred rollback segment is similar to the rollback segment, except that it contains only rollback entries for a given tablespace (from many transactions and instances), where standard rollback segments contain only rollback entries for a given instance (from many transactions and tablespaces).

Temporary Segments

When processing queries, ORACLE often requires temporary work space for intermediate stages of statement processing. These areas are called *temporary segments*. Typically a temporary segment is required as a work area for sorting. A segment is not created if the sorting operation can be done in memory, or if ORACLE RDBMS finds some other way to perform the operation using the indexes.

Operations Requiring Temporary Segments

The following SQL operations may require the use of a temporary segment:

- CREATE INDEX
- SELECT ... ORDER BY
- SELECT DISTINCT ...
- SELECT ... GROUP BY
- SELECT ... UNION
- SELECT ... INTERSECT
- SELECT ... MINUS
- unindexed joins
- certain correlated subqueries.

For example, if a query contains a DISTINCT clause, a GROUP BY, and an ORDER BY, as many as three temporary segments might be required.

How Temporary Segments are Allocated

Temporary segments are allocated as needed during a user session. For example, if a user has three simultaneous queries in progress that require temporary segments, three are allocated. Temporary segments are dropped when the statement completes.

Temporary segments are created in the tablespace specified for the user via ALTER USER, or in the SYSTEM tablespace if none has been specified. The storage characteristics are determined in the usual way, using defaults for the SYSTEM and other tablespaces.

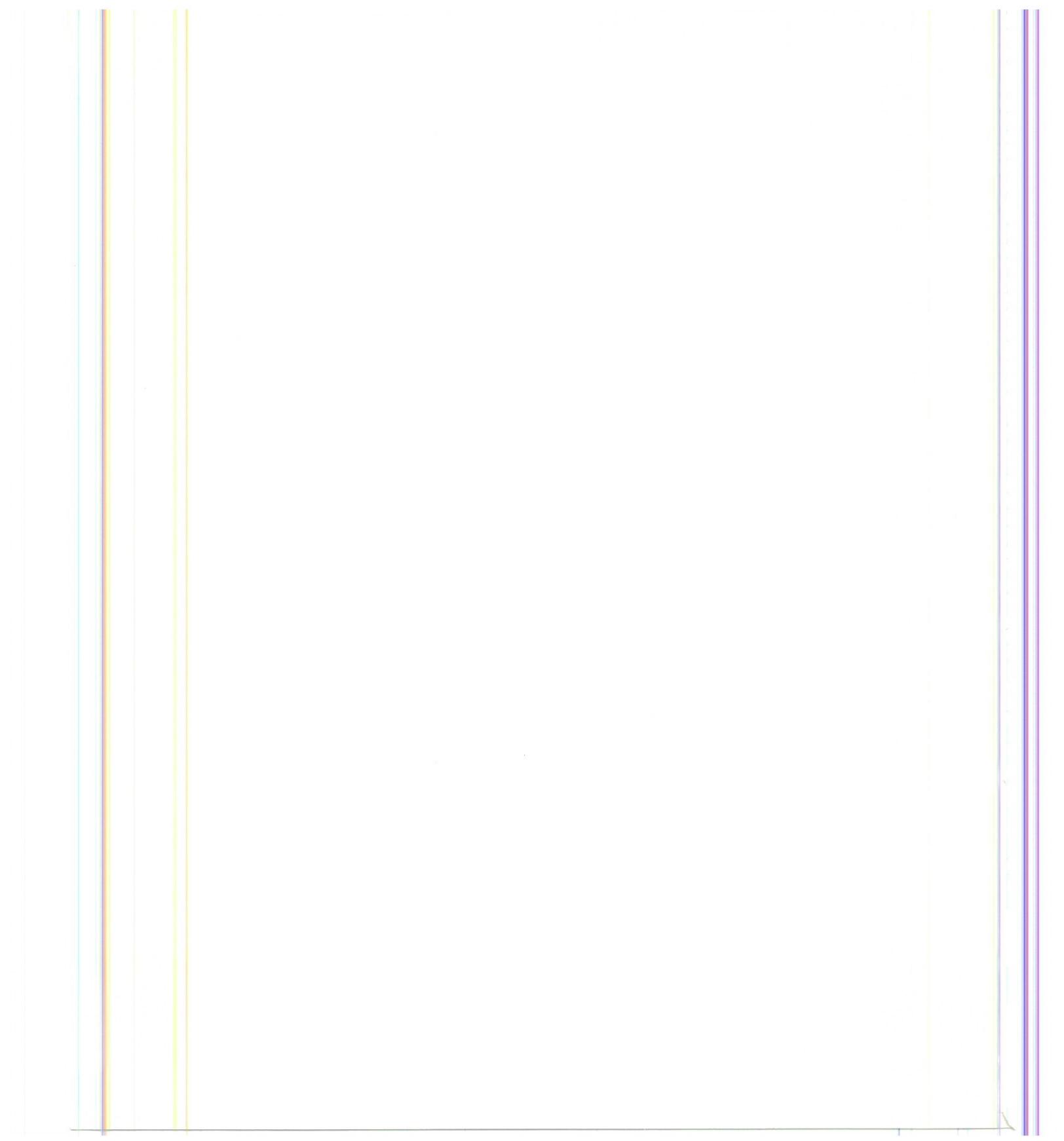
Because allocation and deallocation of temporary segments occur frequently, it is reasonable to create a special tablespace for temporary segments. By doing so, you can distribute I/O across disk devices and you may avoid fragmentation of the SYSTEM and other tablespaces.

The redo log does not contain any entries for changes made to temporary segments.

The Bootstrap Segment

The bootstrap segment is created in the SYSTEM tablespace when a database is created. It contains dictionary definitions for dictionary tables, to be loaded when the database is opened. The bootstrap segment requires no action on the part of the DBA.

This segment is small (usually less than 50 database blocks). Though it remains in the database (inaccessible to users), it does not change in size. A DBA can see evidence of this segment by querying the view DBA_SEGMENTS where SEGMENT_TYPE equals CACHE.



6 ORACLE DATATYPES

Character is simply habit long continued.

Plutarch: *On Moral Virtue*

If you want to be witty, work on your character and say what you think on every occasion.

Stendahl

This chapter covers several topics regarding ORACLE datatypes:

- the definitions and uses of the various datatypes
 - CHAR , VARCHAR (essentially identical)
 - NUMBER
 - DATE
 - LONG
 - RAW
- LONG RAW
- how they are stored, used, and converted
- how they correspond to or differ from other datatypes
- the pseudo-datatype ROWID.

Data in an ORACLE RDBMS is stored as rows in tables. A table may contain as many as 254 columns, and each column is specified as one of the ORACLE datatypes. Descriptions of the individual datatypes follow.

The CHAR and VARCHAR Datatypes

The CHAR and VARCHAR datatypes are equivalent. These datatypes are used to store character (alphanumeric) data. You may use the words CHAR and VARCHAR interchangeably, with the same effect.

Data is stored in variable length strings of ASCII or EBCDIC values, depending on the character set of the host computer. The use of non-standard characters may compromise compatibility across machine architectures.

The absolute maximum number of characters which can be stored in any column defined as CHAR or VARCHAR is 255. The maximum length of a given CHAR or VARCHAR column is specified at table creation and can be altered later.

Columns defined as CHAR or VARCHAR appear in the data dictionary as VARCHAR.

While CHAR and VARCHAR are treated equally in Version 6.0, it is expected that in a future version of the ORACLE RDBMS, CHAR data will be fixed-length, and VARCHAR will be varying character strings up to a declared length.

NUMBER Datatype

The NUMBER datatype is used to store numbers (fixed or floating point). Numbers of virtually any magnitude may be stored, up to 38 digits of precision. Numbers as large as 9.99×10^{124} , or 1 followed by 125 zeroes, can be stored.

For numeric columns you can simply specify NUMBER, as in

```
( colname NUMBER )
```

or you can specify a *precision* (total number of digits) and *scale* (number of digits to right of decimal point)

```
( colname NUMBER ( precision, scale ) )
```

as in:

```
CREATE TABLE MINI (weenumbers number (2,1));
```

You cannot specify a scale greater than 38. You may specify a scale and no precision by specifying:

```
(colname number (*,2) )
```

Using Scale and Precision

When specifying numeric fields, it is a good idea to specify the maximum number of digits and decimal places. This provides extra integrity checking on input. Figure 6-1 shows examples of how data would be stored using different scale factors.

If the scale is negative, the actual data is rounded to the specified number of places, to the left of the decimal point. For example, a specification of (10,-2) means to round to hundreds.

Figure 6-1
Examples of Scale and Precision on NUMBER Data

Actual Data	Specified As	Stored As
7,456,123.89	NUMBER	7456123.89
7,456,123.89	NUMBER (*,1)	7456123.9
7,456,123.89	NUMBER (9)	7456124
7,456,123.89	NUMBER (9,2)	7456123.89
7,456,123.89	NUMBER (9,1)	7456123.9
7,456,123.8	NUMBER (6)	(not accepted, exceeds precision)
7,456,123.89	NUMBER (7,-2)	7456100

Internal Numeric Format

ORACLE uses its own internal numeric format for a variety of reasons including portability and precision (no rounding error). Numeric data is stored in variable length format, starting with an exponent and sign byte, followed by data bytes. A maximum of 21 data bytes can be used (22 including a length byte); each byte contains two decimal digits.

```
[sign/exponent] digit-1 digit-2 digit-3 ... digit-19
```

The sign bit is the high-order bit of the exponent byte. If it is set, the number is positive; if it is clear, the number is negative. The exponent is the low-order seven bits of the exponent byte. The exponent is base-100, can range from 0 to 127, and is in excess-64 notation. The result of 64 subtracted from the exponent tells how many bytes to shift the number, where each shift is a multiply or divide by 100. If the result is negative, then the number is to be shifted to the right of the implied radix point; if the result is positive, then the number is to be shifted to the left.

Each digit represents a binary number from 0 to 99, representing a base-100 digit. Each digit has 1 added to it to prevent binary 0 from occurring as a number. Thus, the actual internal representation of each

digit is a binary number from 1 to 100. The first digit is always non-zero because all numbers are normalized.

The radix point is to the left of the first digit. Negative numbers are transformed positive numbers. This transformation assures that all ORACLE numbers will collate correctly.

The transformation method is as follows. The negative exponent byte, including the sign bit, is the one's complement of the positive exponent byte. Each negative digit is the 100's complement of its corresponding positive digit. This means that a positive digit is subtracted from 100 to get the corresponding negative digit. If a negative number does not have the maximum number of digits then a byte containing 102 is appended to the number.

Zero is a one-byte number containing 128. This is the exponent byte, with the sign bit set and the smallest possible exponent (-64). This guarantees that zero will collate below all positive numbers and above all negative numbers.

Positive infinity is represented by two bytes [255,101]; **negative infinity** is represented by three bytes [1,1,102]. These representations were chosen so the two infinities would collate correctly.

DATE Datatype

Date data is stored in fixed length fields of seven bytes each. For each date the following information is stored:

BYTE:	1 & 2	3	4	5	6	7
	Year	Month	Day	Hour	Minute	Second

The first two bytes stores the year (including the century). BC years are stored in 2's complement form. Time is stored in 24-hour format. By default, the time in a date field is 12:00 p.m. (midnight) if no time portion is entered. Time-only entries default the date portion to the system date.

Standard ORACLE DATE format is DD-MON-YY, as in 13-NOV-88. To enter dates which are not in standard ORACLE date format, use the TO_DATE function with a format mask.

ORACLE can store dates ranging from Jan 1, 4712 BC through Dec 31, 4712 AD. AD is assumed unless a format mask is used to indicate BC.

To enter the time portion of a date, the TO_DATE function must be used with a format mask indicating the time portion, as in:

```
INSERT INTO BIRTHDAYS (BNAME, BDAY) VALUES ('ANNIE',
TO_DATE('13-NOV-85 10:56 A.M.', 'DD-MON-YY HH:MI A.M.')
```

To compare dates which have time data, you can use the SQL function TRUNC to ignore the time component.

Using Julian Dates

The format mask "J" can be used with date functions (TO_DATE or TO_CHAR) to convert date data into Julian dates. For example

```
SELECT TO_CHAR (HIREDATE, 'J') FROM EMP
```

will show all hiredates in Julian date format. The TO_NUMBER function must also be used if you want to use Julian dates in calculations.

Julian dates can be calculated and interpreted differently; the calculation method used by ORACLE RDBMS results in a 7 digit number (for dates most often used), as in 2444993 for 23-JAN-82. The intent of Julian dates is to allow continuous dating from a common reference. (The reference is some 4000 years B.C., so current dates are somewhere in the 2.4 million range.) A Julian date is nominally a non-integer, the fractional part being a portion of a day. The ORACLE RDBMS uses a simplified approach which results in integer values.

LONG Datatype

Columns defined as LONG can store variable length character strings containing up to 65,535 characters. LONG data is simply an unstructured set of characters. LONG data can be used to store arrays of binary data, arbitrary characters, or even short documents.

LONG datatypes are used in the data dictionary to store the text of view definitions. Columns defined as LONG can be used in SELECT lists, SET clause of UPDATE statements and INSERT statements, but do have some restrictions, noted below.

LONG data differs from LONG RAW in that LONG RAW columns are assumed to be non-character data, so character set conversions don't occur when moving data between systems.

Restrictions on LONG Data

Though LONG columns have many uses, there are some restrictions on their use. Only one LONG column is allowed per table. Additionally, LONG columns cannot be:

- indexed
- used in WHERE, GROUP BY, ORDER BY, CONNECT BY, or DISTINCT clauses
- referenced by functions (such as SUBSTR or INSTR)
- used in the SELECT list of nested queries
- used in expressions
- used in the SELECT list of a query block connected with another query block by UNION, INTERSECT, or MINUS
- used in distributed queries (although they can be used in the COPY command of SQL*Plus).

Some ORACLE products may not allow buffers large enough to handle very long LONG data (for example, SQL*Plus can be used to enter LONG data only up to a certain limit).

RAW and LONG RAW Datatypes

The RAW and LONG RAW datatypes are used for byte-oriented data that is to be uninterpreted by ORACLE. RAW is similar to CHAR data (and LONG RAW is like LONG) except that no assumptions are made about the meaning of the bytes.

These datatypes are intended for binary data or byte strings, for example, to store graphics character sequences. RAW data is equivalent to CHAR (and LONG RAW to LONG) except when transmitted by SQL*Net, at which time CHAR is converted to a different character set, if necessary, and RAW is not.

When ORACLE automatically converts RAW or LONG RAW data to and from CHAR data (as is the case when entering RAW data from the keyboard using SQL*Plus) the data is represented as 1 hexadecimal character representing the bit pattern for every 4 bits of RAW data. One byte of RAW data with bits 11001011 would be displayed and entered as 'BA'.

LONG RAW data cannot be indexed.

How NULL Values are Stored

A *null* value is the lack of a value in a column. Its meaning should be "nothing," and should be used to indicate missing, unknown, or inapplicable data. A null value should not be used to imply any other value, such as zero. A column will allow null data unless it was defined at table creation as NOT NULL, in which case no row can be inserted without a value for that column.

Null columns are "stored" in the database if they fall between columns with data values. In these cases they require one byte. Null values that fall at the end of a row (the null data terminates the row) are not stored, but implied. In tables with many columns, the columns more likely to have null values are usually defined last.

Null values are not indexed with one exception. When all values in a cluster key are null then an index entry is made in the cluster index.

DB2 and SQL/DS Datatypes

In addition to ORACLE datatypes, the SQL statements used to create tables and clusters will accept datatypes from IBM's products SQL/DS and DB2, and internally convert them to ORACLE datatypes as follows:

<i>SQL/DS or DB2 datatype</i>	<i>ORACLE datatype</i>
SMALLINT	NUMBER
INTEGER	NUMBER
DECIMAL(m,n)	NUMBER (m,n)
FLOAT	NUMBER
VARCHAR(n)	VARCHAR (n)
LONG VARCHAR	LONG
CHARACTER (n)	CHAR (n)

Because they have no corresponding ORACLE datatype, the SQL/DS datatypes GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC should never be used. However, all SQL/DS datatypes are reserved words, and so cannot be used to name database objects such as tables or indexes.

ANSI Datatypes

ORACLE automatically converts ANSI datatypes to ORACLE datatypes as shown in the following table:

ANSI Datatype	ORACLE Datatype
CHARACTER (n)	CHAR (n)
CHARACTER, CHAR	CHAR (1)
NUMERIC [(p [,s])]	NUMBER [(p [,s])]
DEC [(p [,s])]	NUMBER [(p [,s])]
SMALLINT	NUMBER (*,0)
INTEGER	NUMBER (*,0)
FLOAT [(p)]	NUMBER
REAL	NUMBER (x)
DOUBLE PRECISION	NUMBER

The Pseudo-Datatype ROWID

Associated with every row in the database is a logical column which corresponds to the address of that row. That address can be retrieved with a SQL query using the reserved word ROWID. As the following example shows, a query selecting ROWID will return a result with a hexadecimal representation of the address for each row selected:

```
SELECT ROWID, ENAME FROM EMP;
ROWID          ENAME
-----
00000938.0000.0001 SMITH
00000938.0001.0001 ALLEN
00000938.0002.0001 WARD
```

ROWID returns three pieces of information necessary to locate a row:

- which *block* in the file (block 938 in the example)
- which *row* in the block (first row is 0 — example shows rows 0, 1, and 2)
- which *file* it is in (first file is 1)

Row numbers start with 0. Block numbers are relative to their file, not tablespace. Therefore, two rows with identical block numbers could reside in two different files. File numbers are distinct within a database.

Why Use ROWIDs?

Rowids have several important uses:

- They are the fastest means of accessing particular rows.
- They can be used to see how a table is organized.
- They are unique identifiers for rows in a given table.

Because the ROWID for a given row is not guaranteed to remain constant, (it will change when a row is exported and imported) you should verify a ROWID for a row before using it to manipulate the row.

Though ROWIDs can be referenced like table columns (used in SELECT lists and WHERE clauses), they are not stored in the database, nor are they database data. Thus, it is not possible to UPDATE, INSERT or DELETE a ROWID.

Examples of Using ROWID

Using some group functions with ROWID, you can see how data is internally stored in the ORACLE database.

The function SUBSTR can be used to break the data in ROWID into its three components (file, block, and row):

```
SELECT ROWID, SUBSTR(ROWID,15,4) FILE,
       SUBSTR(ROWID,1,8) BLOCK,
       SUBSTR(ROWID,10,4) ROW
  FROM EMP;
```

ROWID	FILE	BLOCK	ROW
000000A8.0000.0001	0001	000000A8	0000
000000A8.0001.0001	0001	000000A8	0001
000000A8.0002.0001	0001	000000A8	0002
000000A8.0003.0001	0001	000000A8	0003

The following query tells how many of the blocks allocated are actually used by a table's current data:

```
SELECT COUNT(DISTINCT(SUBSTR(ROWID,1,8))) BLOCKS
  FROM tablename
```

Note that this result reflects only data blocks containing the first portion of a row; it does not include:

- the one block required for each table's overhead
- blocks containing only row overflow pieces
- blocks which were allocated in the extent but do not yet have data.

The following query tells how many rows begin in each block belonging to a particular table:

```
SELECT SUBSTR(ROWID,1,8) BLOCK, COUNT(*)  
  FROM tablename  
 GROUP BY SUBSTR(ROWID,1,8)
```

Data Conversion

WHERE clauses and expressions (boolean, arithmetic, or string) may reference different datatypes in their comparisons. In these cases, ORACLE can often use data conversion to resolve the statement. In the following examples, ORACLE automatically performs the implied data conversions:

```
SELECT ENAME FROM EMP WHERE ENAME = 135  
SELECT ENAME FROM EMP WHERE EMPNO = '7936'  
SELECT ENAME FROM EMP WHERE HIREDATE = '12-MAR-86'  
SELECT ENAME FROM EMP WHERE ROWID = '00002514.0001.0001'
```

To perform the conversion, ORACLE can:

- convert the constant to the column's defined datatype
- convert the column's value to the datatype of the constant
- convert one column's datatype to the datatype of another column.

Data conversion is context-dependent. Do not expect the same sort of conversion to work in every case. Rather than rely on implied or automatic conversions, it is best to specify explicit conversions with the SQL functions provided for that purpose. If you leave the conversion up to the ORACLE RDBMS, it may have a negative effect on performance, particularly if a column's datatype is converted to that of a constant rather than the other way around.

Algorithms for conversion are subject to change across software releases and among ORACLE products.

7

THE DATA DICTIONARY

No dictionary of living tongue ever can be perfect, since while it is hastening to publication, some words are budding and some falling away.

Samuel Johnson: Preface to *Dictionary*

This chapter describes the creation and purpose of a central set of reference tables and views known collectively as the *data dictionary*. Topics include:

- the definition and use of the data dictionary
- what types of objects constitute the data dictionary
- the underlying design and naming scheme
- creating and altering the data dictionary.

Not only is the data dictionary a central part of the ORACLE RDBMS software, it is an important tool for all users of ORACLE, from casual users to applications designers and DBAs. Even beginning users will benefit from understanding the data dictionary's many uses as early as possible.

Note: No user should ever alter (update, delete, or insert) any rows or objects in the data dictionary. Such activity has a strong chance of compromising data integrity.

While reading this chapter, you may find it helpful to refer to Appendix E, which contains descriptions of the individual data dictionary tables and views. The *Installation and User's Guides* describe installation steps relating to the creation of the data dictionary.

What is the Data Dictionary?

One of the most important parts of the ORACLE RDBMS is the data dictionary. The *data dictionary* is a set of tables to be used as a **read-only** reference guide about the database. For example, it will tell you:

- the usernames of ORACLE users
- rights and privileges they have been granted
- names of database objects (tables, views, indexes, clusters, synonyms, and sequences)
- information about primary and foreign keys
- default values for columns
- constraints applied to a table
- how much space has been allocated for, and is currently used by, the objects belonging to a database user
- auditing information, such as who has accessed or updated various database objects
- other general database information.

The data dictionary is structured in tables and views, just like other database data. To access the data dictionary, you use the SQL language. Because the data dictionary is read-only, you will use various queries (SELECT statements). Because many views of the data dictionary are accessible to all database users, new users can learn SQL simply by forming queries against the data dictionary.

The data dictionary is created when a database is created. Thereafter, whenever the database is in operation, the data dictionary is updated by the ORACLE RDBMS in response to every DDL statement. At all times it is available as a reference for any user, whether or not that user has created any database objects.

The data dictionary is a valuable source of information for end users, application designers, and DBAs alike. It is also critical for the operation of the ORACLE RDBMS, which relies on the data dictionary to record, verify, and conduct ongoing work.

The View DICTIONARY

The view DICTIONARY lists the data dictionary objects with a brief description. If you forget the name of the table you want, always start with DICTIONARY.

Note the column COMMENTS in the view DICTIONARY; when the new data dictionary objects are created, comments are loaded for every view and column, providing online documentation explaining these objects and columns.

Access to the Data Dictionary

The data dictionary is a reference for all database users. All database users have equal access to any data dictionary view beginning with USER or ALL.

Some data dictionary objects are for DBAs only. Those views are prefixed with DBA and cannot be accessed by normal users.

The data dictionary is always available when the database is open. It resides in the SYSTEM tablespace, which must always be online when the database is open.

Access to the data dictionary objects is via the SQL language.

The Data Dictionary Tables and Views

A database is created with two DBA users, SYS and SYSTEM (see Chapter 2). SYS owns the base (or underlying) data dictionary tables and the views which summarize the data in the base tables.

Note: No ORACLE user should ever alter an object belonging to SYS with one exception (see "Deleting Data Dictionary Items").

The views contain information in a more convenient form than is provided by the base tables. The base tables are rarely accessed directly because they are normalized, and most of the data is stored using numeric keys rather than easily understood names. The views decode this information into their meanings, such as user or table names, and use joins and WHERE clauses to simplify the information. Thus, public access to the data dictionary is given on views rather than the base tables.

The Data Dictionary Structure

The data dictionary consists of sets of views. Each set contains a logical set of information; for example, there are views corresponding to every type of database object which may be created by users or DBAs.

In many cases a set consists of three views containing similar information and distinguished from each other by their prefixes:

Prefix	Scope
USER	User's view (what a user owns)
ALL	Expanded user's view (what a user can access)
DBA	DBA's view (summary of all users)

The set of columns is basically identical across views with some exceptions:

- Views with the prefix USER usually exclude the column OWNER. This column is included in the ALL and DBA views, but is implied in the USER views to be the username running the query.
- Some DBA views have additional column(s) containing information pertinent only to the DBA.

Views with the Prefix USER

The views most likely to be of interest to normal database users are those with the prefix USER. These views:

- refer to the user's own private environment in the database; that is, they display information on tables created by the user, grants made by the user
- display the fewest rows (and therefore, the most manageable)
- have basically identical columns to the other views, except that the column OWNER is implied (the current username)
- return a subset of what will be returned by the views with the prefix ALL
- may have abbreviated PUBLIC synonyms for convenience.

View with the Prefix ALL

Views with the prefix ALL refer to the user's overall perspective of the database. These views return information about database objects to which the user has access via public or explicit grants, in addition to

objects he owns. If a user with DBA privilege queries these views, he will see only objects to which he's been given explicit access.

Views with the Prefix DBA

Views with the prefix DBA may be queried only by DBAs because they show a global view of the entire database. No synonyms are created for these views, as the views should only be queried by the DBA. Thus, to query these views DBAs must use the prefix SYS before the view name, as in:

```
SELECT * FROM SYS.DBA_TABLES
```

Alternatively, a DBA can run the file DBA_SYNONYMS.SQL to create synonyms for these views; the execution of this file creates synonyms available to the current user only, assuming that user has DBA privilege.

ANSI Compatible Views

Several views have an alternate name provided for ANSI compatibility. In each case, one view and a synonym for the view are created. Those views and their synonyms are:

<i>View Name</i>	<i>ANSI Synonym</i>
ALL_CATALOG	ACCESSIBLE_TABLES
ALL_TAB_COLUMNS	ACCESSIBLE_COLUMNS
USER_USERS	MYPRIVS
ALL_TAB_GRANTS	TABLE_PRIVILEGES
ALL_COL_GRANTS	COLUMN_PRIVILEGES

Other Views

Some data dictionary views do not use the prefixes USER, ALL, or DBA. Those views include:

DICTIONARY lists all data dictionary tables, views, and synonyms accessible to the current user.

DICT_COLUMNS lists descriptions of columns in DICTIONARY objects accessible to the current user.

CONSTRAINT_DEFS lists all constraint definitions entered for tables that are accessible to the current user.

CONSTRAINT_COLUMNS lists all columns that are accessible to the current user and named in constraint definitions

AUDIT_ACTIONS lists all operations that may be audited and their corresponding code.

DUAL is a small table referenced by ORACLE and user-written programs to guarantee a known result. It has one column and one row.

User's Own Objects	(Synonym)	Expanded User View	DBA's Global View
--------------------	-----------	--------------------	-------------------

General

(DICT)	DICTORY DICT_COLUMNS CONSTRAINT_DEFS CONSTRAINT_COLUMNS AUDIT_OPTIONS DUAL
--------	-------------------------------------------------------------------------------------------

User Objects

USER_CATALOG	(CAT)	ALL_CATALOG or ACCESSIBLE_TABLES	DBA_CATALOG
USER_OBJECTS	(OBJ)	ALL_OBJECTS	DBA_OBJECTS
USER_TAB_COMMENTS		ALL_TAB_COMMENTS	DBA_TAB_COMMENTS
USER_COL_COMMENTS		ALL_COL_COMMENTS	DBA_COL_COMMENTS
USER_CROSS_REFS			DBA_CROSS_REFS
USER_TABLES	(TABS)	ALL_TABLES	DBA_TABLES
USER_TAB_COLUMNS	(COLS)	ALL_TAB_COLUMNS or ACCESSIBLE_COLUMNS	DBA_TAB_COLUMNS
USER_INDEXES	(IND)	ALL_INDEXES	DBA_INDEXES
USER_IND_COLUMNS		ALL_IND_COLUMNS	DBA_IND_COLUMNS
USER_CLUSTERS	(CLU)		DBA_CLUSTERS
USER_CLU_COLUMNS			DBA_CLU_COLUMNS
USER_VIEWS		ALL_VIEWS	DBA_VIEWS
USER_SYNONYMS	(SYN)	ALL_SYNONYMS	DBA_SYNONYMS
USER_SEQUENCES	(SEQ)	ALL_SEQUENCES	DBA_SEQUENCES
USER_DB_LINKS		ALL_DB_LINKS	DBA_DB_LINKS

Users

USER_USERS or MYPRIVS	ALL_USERS	DBA_USERS
USER_TS_QUOTAS		DBA_TS_QUOTAS

User's Own Objects**Expanded User View****DBA's Global View*****System Objects***

USER_TABLESPACES

DBA_DATA_FILES

DBA_TABLESPACES

DBA_ROLLBACK_SEGS

Storage

USER_SEGMENTS

DBA_SEGMENTS

USER_EXTENTS

DBA_EXTENTS

USER_FREE_SPACE

DBA_FREE_SPACE

Grants

USER_TAB_GRANTS

ALL_TAB_GRANTS
or TABLE_PRIVILEGES

DBA_TAB_GRANTS

USER_COL_GRANTS

ALL_COL_GRANTS
or COLUMN_PRIVILEGES

DBA_COL_GRANTS

USER_TAB_GRANTS_REC'D

ALL_TAB_GRANTS_REC'D

USER_COL_GRANTS_REC'D

ALL_COL_GRANTS_REC'D

USER_TAB_GRANTS_MADE

ALL_TAB_GRANTS_MADE

USER_COL_GRANTS_MADE

ALL_COL_GRANTS_MADE

Auditing

USER_TAB_AUDIT_OPTS

DBA_TAB_AUDIT_OPTS

ALL_DEF_AUDIT_OPTS

USER_AUDIT_TRAIL

DBA_AUDIT_TRAIL

USER_AUDIT_CONNECT

DBA_AUDIT_CONNECT

USER_AUDIT_RESOURCE

DBA_AUDIT_RESOURCE

DBA_AUDIT_DB

DBA_AUDIT_EXISTS

DBA_SYS_AUDIT_OPTS

Export/Import

DBA_EXP_VERSION

DBA_EXP_FILES

How is the Data Dictionary Created?

The data dictionary base tables must be the first objects to be created in any database, as they must be present for all other objects to be created. Data dictionary tables are automatically created by the SQL statement `CREATE DATABASE`.

During database creation the `INIT.ORA` parameter, `INIT_SQL_FILES`, is read to find the name of one or more files which should be run (the default filenames vary by operating system). The files must always create the data dictionary first, but other files can also be run to create initial DBA or site-specific tables. For example, you could add names of your files after the default values, as in:

```
INIT_SQL_FILES = (SQL.BSQ, CATALOG.ORA, ACME_DBA.SQL)
```

If you use additional files and they create objects, the objects should belong to either `SYSTEM` or a newly created DBA username, not `SYS`.

The initial SQL files:

1. Define the `SYSTEM` tablespace and `SYSTEM` rollback segment.
2. Define the data dictionary base tables.
3. Load data into some data dictionary tables.
4. Define the data dictionary views.
5. Create public synonyms for the many of the views.
6. Grant `PUBLIC` access to those synonyms.

The data dictionary is created in and must remain in the `SYSTEM` tablespace. For every table and column in a base table a comment is also loaded, to provide online documentation.

Public synonyms are created on the non-DBA views so all ORACLE users can access the views conveniently. In many cases, short public synonyms are also created for quick entry.

How is the Data Dictionary Used?

Data in the base tables of the data dictionary is not only useful for users and DBAs, but is necessary for the RDBMS to function. Thus, only the RDBMS should write or change data dictionary information.

During database operation the RDBMS reads the data dictionary to ascertain that database objects exist and that users have proper access to them. The RDBMS also updates the data dictionary continuously to reflect changes in database structures, auditing, grants, and data.

For example, if user NANCY creates a table named PARTS, new rows are added to reflect the new table, columns, segment, extents, and privileges NANCY has on the table. These changes are then visible the next time the dictionary views are queried.

Caching of the Data Dictionary for Fast Access

Because the data dictionary is accessed by the ORACLE RDBMS so constantly during database operation, to validate user access and to verify the state of database objects, much of the data dictionary information is cached in the SGA. All information is stored using the LRU (least recently used) algorithm.

The amount of memory used for dictionary caches is determined by the set of INIT.ORA parameters beginning with DC (for dictionary cache). See Appendix D for a discussion of these parameters as a group and individually.

Information typically kept in the caches is that required for parsing. The COMMENTS columns describing the tables and columns are not cached unless they are frequently accessed.

Other Programs and the Data Dictionary

Other ORACLE products may either create additional data dictionary tables or views of their own, or reference existing views. If you write programs that reference the data dictionary, your program should reference the public synonyms, rather than the underlying tables, as the synonyms are less likely to change between software releases.

Adding New Data Dictionary Items

It is permissible to add entirely new tables or views to the data dictionary. If you add new data dictionary objects, the owner of the new objects should be the DBA user SYSTEM or a third ORACLE user with DBA privileges. Do not create new objects belonging to user SYS.

Deleting Data Dictionary Items

Because all changes to the data dictionary are performed by the database itself in response to DDL statements, **no data in any data dictionary tables should be deleted or altered by any user.**

The single exception to this is the table SYS.AUD\$. Depending on which audit options are in effect, this table may grow without bound. Though you should not DROP the AUDIT_TRAIL table, you may delete data from it. The rows are for information only and are not necessary for the RDBMS to run.

Public Synonyms for Data Dictionary Views

Public synonyms are created on many data dictionary views so they can be conveniently accessed by users. It is fine to create additional public synonyms for objects used system-wide. However, you should avoid creating your own objects with the same names as those used for public synonyms.

The Dynamic Performance Tables

Throughout its operation, the ORACLE RDBMS maintains a set of "virtual" tables that record current database activity. These tables are called *dynamic performance tables* (or sometimes, *fixed tables*). Information in these tables is used in several of the MONITOR displays.

Because dynamic performance tables are not true database tables, they should not be accessed by most database users. However, DBAs can query these tables, and may create views on the tables and grant access to those views to database users.

The dynamic performance tables are owned by SYS and their names all begin with V_\$. In all databases, views are created on these tables, and then synonyms are created for the views. The synonym names begin with V\$.

Appendix E contains a listing of the synonyms and their columns and contains more information about these tables.

8

ORACLE MEMORY STRUCTURES

Own only what you can always carry with you; know language, know countries, know people. Let your memory be your travel bag.

Alexander Solzhenitsyn

When I was younger I could remember anything, whether it had happened or not.

Mark Twain

This chapter describes the various ways ORACLE uses memory, including user areas and global areas. Topics include:

- general descriptions of how ORACLE uses memory
- the System Global Area (the SGA)
- Program Global Areas (the PGA)
- context areas
- software areas.

For more detailed information about memory usage and requirements, refer to the *Installation and User's Guide* for your operating system. The INIT.ORA parameters described in Appendix D also affect the memory requirements of an ORACLE database system.

How ORACLE Uses Memory

ORACLE RDBMS uses memory for the following purposes:

- to store program code being executed
- to store data needed during program execution (for example, the current state of a query from which rows are being fetched)
- to store information which is shared and communicated among ORACLE processes (for example, locking information)
- to store information which is transferred between processes and peripheral memory (for example, a database block being transferred from disk to program).

On many operating systems, ORACLE takes advantage of *virtual memory*. Virtual memory is an operating system feature which offers more apparent memory than is provided by real memory alone and more flexibility in using main memory.

Virtual memory simulates memory using a combination of real (main) memory and secondary storage (usually disk space). The operating system accesses virtual memory by making secondary storage look like main memory to application programs.

Software Areas

Software areas are portions of memory used to store code that is being or may be executed. The code for the RDBMS is stored in a software area, which is typically at a location different than users' programs — a more exclusive or protected location.

Size

Software areas are usually static in size, only changing when the software is updated or re-installed. The size required varies by operating system.

Read-Only, and Shared or Non-shared

Software areas are read-only and may be installed shared or non-shared. (Installing software shared is not an option for all operating systems; for example, it is not on PCs.) When possible, the RDBMS code is shared so that all ORACLE users can access it without having multiple copies in memory. This results in a saving of real main memory, and usually better overall system performance.

User programs can be shared or non-shared; some ORACLE utilities are installed shared and some are not. Multiple ORACLE instances can use the same ORACLE code area if running on the same CPU.

System Global Area (SGA)

The System Global Area (usually called the SGA) contains data and control information for one ORACLE instance. You may hear the SGA referred to as either the System Global Area or the Shared Global Area; both terms are equivalent.

The SGA is allocated when an instance starts and deallocated when the instance is shut down. Current users of the database need to see current copies of information in the SGA.

Because the SGA contains information that is so frequently used, it is best (fastest) to keep the SGA in memory rather than read it from disk. For example, the SGA contains:

- information communicated between processes (such as locking information)
- database buffers
- data dictionary information (stored in dictionary caches).

For optimal performance in most systems, the entire SGA should be able to fit in real memory. If the entire SGA does not fit, then overall database system performance may decrease dramatically, as portions of the SGA are paged in and out by the operating system.

Size

The size of the SGA is determined at instance startup; after that it does not change. To see the size of the SGA after an instance has been started, use the SQL*DBA command SHOW SGA (See Appendix B).

The primary determinant of the size of the SGA is the set of variable parameters found in the INIT.ORA file. Refer to Appendix D for discussions of the parameters, their purposes, and how they affect the SGA. Also refer to your *Installation and User's Guide* for additional information specific to your operating system.

Shared and Writable

The SGA is both shared and writable. It is written only by the RDBMS code, not by user code. One SGA is allocated per instance.

Program Global Area (PGA)

The Program Global Area (usually called the PGA) contains data and control information for a single process, such as a user process. You may hear the PGA referred to as the Program Global Area or the Process Global Area; both terms are equivalent.

A PGA is usually allocated when a user connects to ORACLE, though this varies by operating system. The contents of a PGA include information about the connection (how to communicate with the user program) and what the process is currently doing in ORACLE.

Size

The PGA's size is variable in length but not dynamic. The size is operating system specific. The PGA is allocated at connect time; if sufficient memory is not available to connect then an error will occur. The error will be an ORACLE error in an operating system error number range. A user does not run out of PGA space; rather there is either enough or not enough memory to connect.

The following INIT.ORA parameters have an impact on the size of the PGA for user processes:

- OPEN_CURSORS
- OPEN_LINKS
- SAVEPOINTS
- DB_FILES
- LOG_FILES.

The size of the PGA for the background processes (DBWR, LGWR, SMON, PMON, and ARCH) is affected by some additional parameters.

Non-Shared and Writable

The PGA is a writable non-shared memory area. One PGA is allocated for each connected ORACLE user process. The PGA is exclusive to the user process and is read and written only by the ORACLE code acting on behalf of the user.

Context Areas

Every SQL statement issued by a user process requires a context area. A *context area* is a portion of memory separate from the PGA which contains the state of one SQL statement. The context area contains all data structures needed to execute the SQL statement, including (but not limited to):

- the text of the SQL statement
- a translated form of the SQL statement
- one row of the result and some intermediate values
- cursor status information used to execute the statement
- control information used for sorting.

The term "context area" is often used synonymously with the term *cursor*. See Chapter 10 for a discussion of the distinction between cursors and context areas.

Number of Cursors

A user process can have as many open cursors (and thus context areas) as are set by the INIT.ORA parameter OPEN_CURSORS. A large number of large cursors works best on virtual memory machines; on real memory machines, users should close unneeded cursors.

If a cursor cannot be opened due to a limit on the number of cursors, you can:

- increase OPEN_CURSORS
- increase CONTEXT_AREA
- increase CONTEXT_INCR

Many statements require *recursive cursors* to perform SQL statements on their behalf. For example, a CREATE TABLE causes many updates to various data dictionary tables, to record the new table and columns. *Recursive calls* are made for those recursive cursors; one cursor may execute several calls.

Cursor Size

Context areas start at a given size and grow dynamically if more space is needed. Thus, you will rarely need to be concerned with context space.

The amount of space initially allocated for a context area is set by the INIT.ORA parameter CONTEXT_AREA. The default is 4K for most operating systems. User-written precompiled programs can specify a different initial size when opening cursors.

If more space is needed by the cursor, it is allocated in extents of sizes set by the INIT.ORA parameter CONTEXT_INCR. Up to 50 extents can be allocated.

If a cursor cannot allocate additional memory, an error occurs. To resolve the error, you can:

- increase the operating system per process memory limits
- change the application to use fewer cursors
- close unneeded cursors
- increase CONTEXT_AREA in INIT.ORA
- increase CONTEXT_INCR in INIT.ORA
- increase the size of the cursor when it is opened in the program.

Once storage has been allocated to a cursor, it is available for subsequent statements parsed using that particular cursor. It is not freed until the cursor has been closed. By closing and opening a cursor, the cursor is reset to its original size.

Non-Shared and Writable

Context areas are writable, non-shared memory areas, as they contain information specific to one cursor created by a process. There may be as many allocations as OPEN_CURSORS allows for each process requiring cursors.

9

ORACLE PROCESS STRUCTURE

Solitude is fine but you need someone to tell you that solitude is fine.
Honore de Balzac

This chapter describes the various processes which make up an ORACLE system. Several terms and concepts are introduced, including:

- user processes
- background processes
- single task and two-task ORACLE systems
- multiple-process and single-process systems
- the program interface
- communication protocol drivers.

This chapter provides a general description of the types of processes an ORACLE RDBMS uses; actual implementation and options vary by operating system. For more specific information, refer to the *Installation and User's Guide* for your operating system; for more information on program interface drivers, refer to the *SQL*Net User's Guide*.

What is a Process?

A *process* is a "thread of control" in an operating system; that is, it is a mechanism in an operating system that can execute a series of steps. Some operating systems use the terms *job* or *task*. A process normally has its own private memory area in which it runs.

The process structure of a system (for example, the ORACLE RDBMS) is important because it defines how multiple activities can occur and how they are accomplished. For example, two goals of a process structure might be:

- to simulate a private environment for multiple processes to work simultaneously, as though each process has its own private environment
- to allow multiple processes to share computer resources, which each process needs, but no process needs for long periods of time.

Thus, sharing resources can improve performance.

Types of ORACLE Processes

An ORACLE system has two general types of processes:

user processes	User processes are directly used by a user or application, which perform that user's activities. Every user connecting to the ORACLE database requires a user process.
server processes	Server processes are invoked by processes, which perform functions on behalf of the invoking process.

In turn, there are two types of server processes:

shadow processes	Shadow processes are described later in this chapter in "Two-Task ORACLE and Shadow Processes." Each shadow process is owned by a user process (thus, the term "shadow").
background processes	A standard set of processes (ARCH, DBWR, LGWR, PMON, and SMON) used by multiple-process ORACLE systems, they are described in "The ORACLE Background Processes" later in this chapter. Background processes are not owned by any one user process, but instead perform functions for all users of a database

system. Background processes are started when an instance starts up and end when an instance shuts down.

Single-Task and Two-Task ORACLE

The ORACLE RDBMS process structure has two main variations, depending on whether the user program (or process) and ORACLE program run as the same or separate processes.

Some operating systems offer a choice of single-task or two-task; see your *Installation and User's Guide* for more details on your options.

User Processes

User processes exist in every database system. A user process is invoked by a user running an application program, such as a Pro*C program or a tool such as SQL*DBA. A user process is normally created when a user connects to ORACLE at a terminal or starts a batch job.

User processes communicate with the database through the *program interface*, described later in this chapter.

Single -Task ORACLE

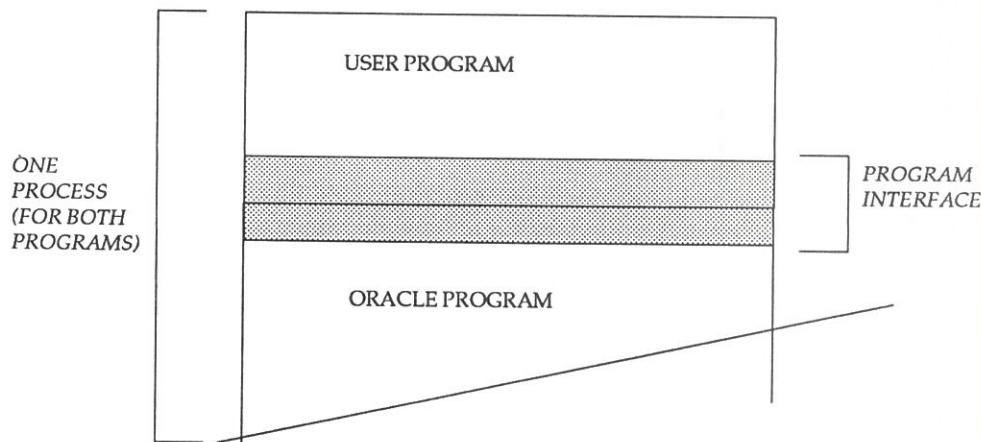
In *single task* systems, the user program, the ORACLE kernel program, and the program interface all run in the same process. In a single-task system, the program interface is responsible for the separation and protection of the ORACLE code and is responsible for passing data between the user and ORACLE.

Single-task ORACLE is feasible in operating systems that can maintain a separation between the user's program and the ORACLE program in a single process. This separation is required for data integrity and privacy. Some operating systems cannot provide this separation and thus must run two-task ORACLE to prevent damage to ORACLE by the user program.

Only one ORACLE connection is allowed at any time by a process using the single-task interface. In a user-written program it is possible, however, to maintain this single task connection while concurrently connecting to ORACLE using a network (two-task) interface.

Figures 9-1 shows the connection between the user process and the ORACLE program in single-task ORACLE.

FIGURE 9-1
Single-Task on
a Single Computer



Two-task ORACLE and Shadow Processes

Two-task ORACLE places the ORACLE process in a separate process from the user process. As in single-task, the program interface performs the communication between the two programs. However, in two-task systems it uses the host operating system's interprocess communication mechanism.

The separate process created to run on behalf of the user's program is called a *shadow process*. In a two-task system, every user process connected to ORACLE has one shadow process.

Figure 9-2 shows a two-task system contained on one CPU.

Two-task ORACLE is also used by SQL*Net, when a user's program is running on a local computer and the ORACLE program and database are located on a remote computer. As shown in Figure 9-3, the program interface also encompasses the communication between the programs, although the protocol and communications links are operating system and installation dependent.

Two-task can also be used in some applications where multiple ORACLE connections can be used with a single database. Single task systems are sometimes provided for debugging or single-user purposes.

Figure 9-2
Two -Task on One Computer

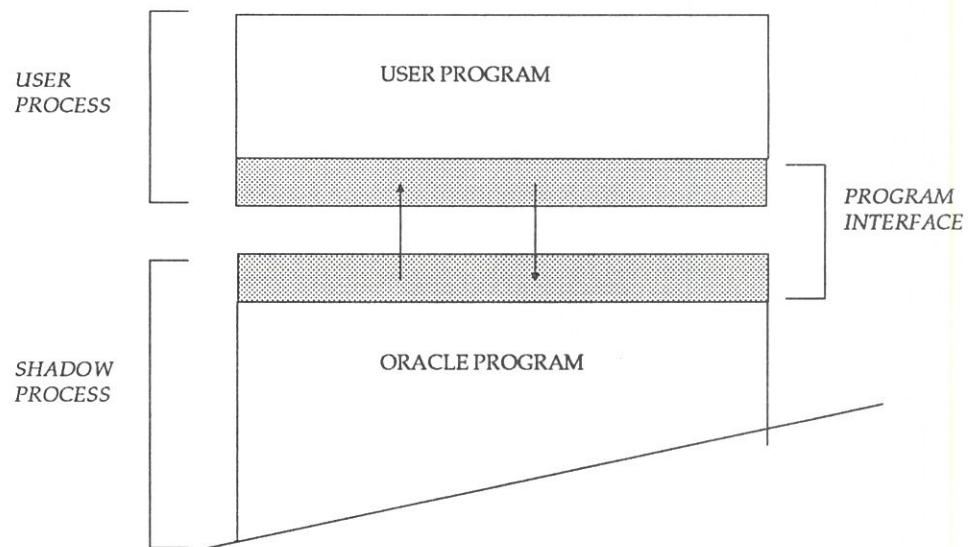
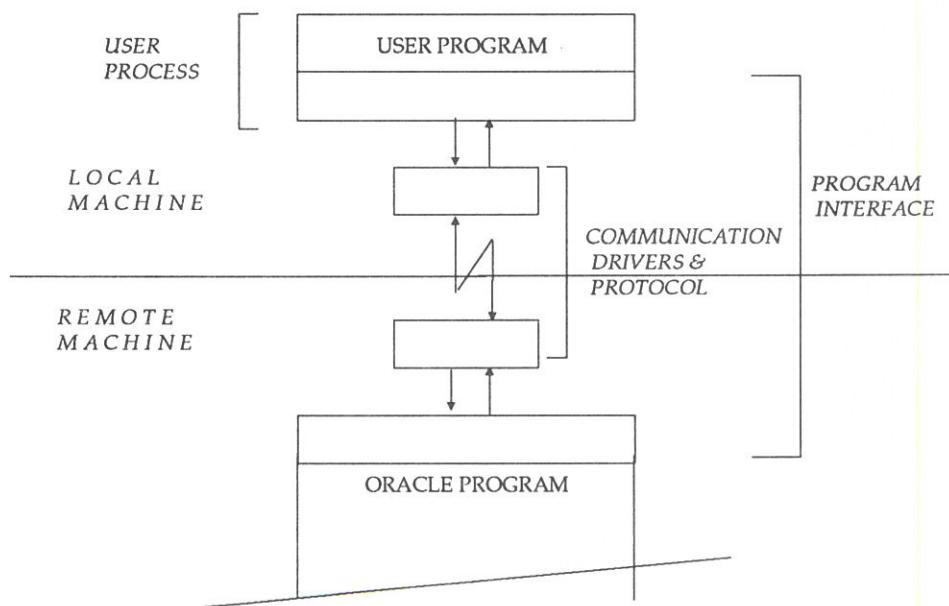


FIGURE 9-3
Two -Task on Two Computers



Multiple-Process and Single-Process Instances

An ORACLE RDBMS can be either single-process or multiple-process. If the value for the INIT.ORA parameter SINGLE_PROCESS is TRUE, then the system is single-process; if it is FALSE then it is multiple-process.

On most operating systems an ORACLE instance can be started either single-process or multiple-process, independent of how it was installed or last started. However, in systems that do not support multiple processes or shared memory, ORACLE can run only as single process (the primary example is ORACLE on MS-DOS based personal computers).

Single-Process ORACLE

Single-process ORACLE (also called single user) is a database system which can be accessed by one user only; multiple users cannot access the database concurrently. ORACLE running on personal computers is single-user, because the MS-DOS operating system is not capable of running multi-user. Single-process is also single-instance; the database cannot be opened by other instances.

Single-process ORACLE instances do not use the five background processes which are used by multi-process ORACLE. Performance of single-process ORACLE is normally slower than multiple-process ORACLE.

Multiple-Process ORACLE

A *multiple-process* ORACLE system (or multi-user) can be used by more than one user or application. Most databases are multi-user, because one of the primary benefits of a database is managing data needed by multiple users at the same time.

A multiple-process system requires up to five background processes plus one process for each database user.

The ORACLE Background Processes

To accommodate many users, a multi-process ORACLE system uses some additional processes called *background processes*. Their purpose is to improve performance by consolidating some functions that would be handled by multiple ORACLE programs running for each user program. The background processes asynchronously perform database reads and writes, and monitor ORACLE processes, to provide greater parallelism for better database performance and reliability.

Each ORACLE instance may use up to five background processes. The names of these processes are DBWR, LGWR, SMON, PMON, and

ARCH. If multiple instances share a database, each instance has its own set of background processes.

On many operating systems these processes are created automatically when an instance is started. On other operating systems, the server processes are like user accounts or IDs and are created as a part of the ORACLE installation. See your *Installation and User's Guide* for details on the processes.

Because it may be desirable to have many ORACLE systems running concurrently on one computer, ORACLE provides a mechanism for naming an instance. The background process names are prefixed by an instance identifier to distinguish the set of processes for each instance. For example, an instance named BIRD running under DEC VMS would have detached processes named:

- ORA_BIRD_DBWR
- ORA_BIRD_LGWR
- ORA_BIRD_SMON
- ORA_BIRD_PMON
- ORA_BIRD_ARCH

Database Writer (DBWR)

The purpose of Database Writer is to write modified blocks from the database buffer cache to the database. Blocks are written in the proper order to maintain database integrity. The algorithm for writing assures that buffers are always available for writing. If additional buffers are needed, older buffers are usually written to the database first.

Because of the way ORACLE performs logging (see Chapter 15), DBWR does not need to write blocks when a transaction commits. The integrity of the data is guaranteed by the writing of committed data in the online redo log, rather than by the writing of data to the database at commit time.

Log Writer (LGWR)

The Log Writer process writes redo log entries to disk. Redo log information is generated in a buffer in the SGA. As transactions commit, LGWR writes redo log entries into the online redo log file. (Sometimes, if more buffer space is needed, LGWR writes redo log entries before a transaction is committed; these entries become permanent only if the transaction is later committed.) LGWR insures a smooth flow of data from buffer to disk without delaying processes that are generating new redo log entries.

System Monitor (SMON)	<p>The purpose of the System Monitor is to perform instance recovery. It is used at instance startup and to perform appropriate recovery for a failed CPU in a shared disk system. SMON is also responsible for cleaning up temporary segments that are no longer in use and for recovering dead transactions skipped during crash and instance recovery because of file-read or offline errors. These transactions are eventually recovered by SMON when the tablespace or file is brought back online.</p> <p>SMON "wakes up" regularly to check whether it is needed, and can be called if another process detects the need for SMON.</p>
Process Monitor (PMON)	<p>The purpose of the Process Monitor is to perform process recovery when a user process fails. PMON is responsible for cleaning up the cache and freeing resources that the process was using. For example, it resets the status of the active transaction table, releases locks, removes the process ID from the list of active ORACLE processes.</p> <p>Like SMON, PMON "wakes up" regularly to check whether it is needed, and can be called if another process detects the need for it.</p>
Archiver (ARCH)	<p>The Archiver process copies the online redo log files to tape when they are full. ARCH is active only when the redo log is used in ARCHIVELOG mode and archiving is enabled (either automatic or manual). For information on enabling archiving refer to Chapter 15.</p> <p>Details of using ARCH are operating system specific, and may require the use of a dedicated tape drive. See Chapter 15 and your <i>Installation and User's Guide</i> for details on using ARCH.</p>

The Program Interface

The *program interface* is the interface between user programs and the ORACLE program. Its functions are to:

- provide a security barrier, preventing destructive access to the database by user processes
- act as a communications mechanism, by formatting information requests, passing data, and trapping and returning errors
- perform conversions and translations of data, particularly between different types of computers or to external user program datatypes.

The *ORACLE* program performs database tasks on behalf of *user programs*, for example, fetching rows from data blocks. It consists of several parts, provided by both *ORACLE* software and operating system specific software.

Program Interface Structure

As shown in Figure 9-4, the program interface consists of the following pieces:

- the *user side* of the program interface (also called the UPI)
- the *ORACLE side* of the program interface (also called the OPI)
- various *drivers*, (protocol-specific communications software)
- operating system communications software
- *ORACLE* call interface (OCI)
- precompiler SQL interface (SQL).

Both the user and *ORACLE* sides of the program interface are *ORACLE* software, as are the drivers.

*SQL*Net* is the portion of the program interface that allows the user program and the *ORACLE* program to exist on separate CPUs.

The Program Interface Drivers

Drivers are concerned strictly with the transport of data. They perform operations like connect, disconnect, signal errors, and test for errors. Drivers are specific to a *communications protocol driver*.

A default driver always exists. The *single-task driver* exists for most *ORACLE* installations and is the default driver.

You may install multiple drivers (such as the asynchronous or DECnet drivers), and select one default driver, but allow an individual user to use other drivers by specifying the desired driver at the time of connection. Different processes can use different drivers. A single process can have concurrent connections to a single database or to multiple databases (either local or remote) using *SQL*Net* drivers.

The *Installation and User's Guides* contain details about choosing and installing drivers and adding new drivers after installation. The *SQL*Net User's Guide* describes selecting a driver at runtime while accessing *ORACLE*.

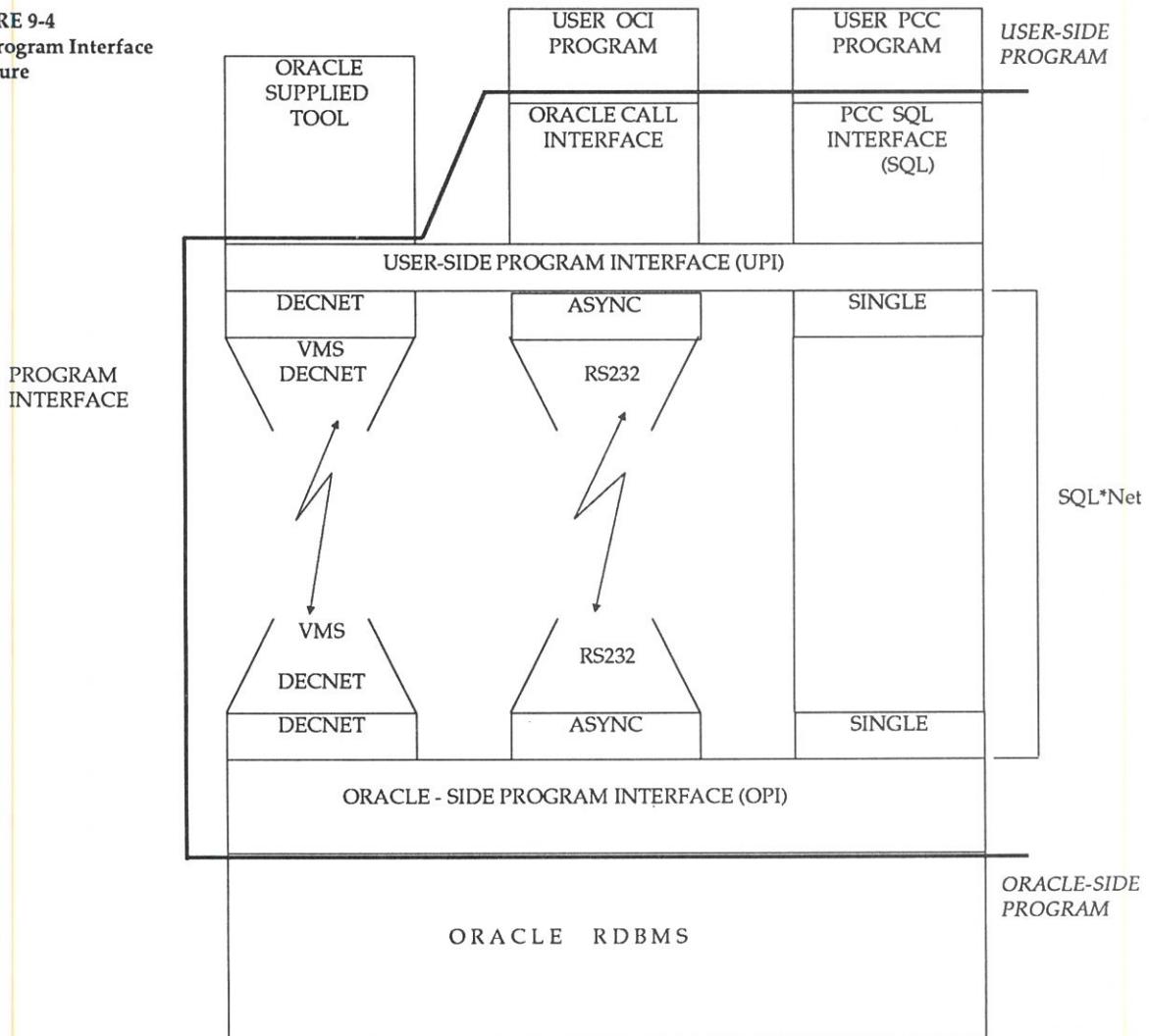
Operating System Communications Software

The ultimate link connecting the user side to the *ORACLE* side of the program interface is the communications software, which is provided by the host operating system. DECnet, TCP/IP, and ASYNC are examples. This link must provide protection (separation) between the

user and operating system halves. The ORACLE-provided drivers are built specifically by Oracle Corporation for each protocol.

The communication software may be supplied by Oracle Corporation but is usually purchased separately from the hardware vendor or a third party software supplier.

FIGURE 9-4
The Program Interface Structure



10

SQL STATEMENT PROCESSING

Perhaps of all the creations of man language is the most astonishing.

Lytton Strachey

A clear statement is the strongest argument.

English proverb

This chapter describes what happens when you issue a SQL statement. Much of your interaction with database data will follow conventions described in this chapter. Topics include:

- definition of SQL statements
- using cursors and context areas
- steps of executing SQL statements:
 - create cursor
 - parse statement
 - bind variables
 - describe results
 - define output
 - execute
 - for a query, fetch rows of result
- special concerns of queries.

Although some ORACLE tools and applications simplify or mask the use of SQL, SQL statements provide the only means of selecting or altering data in the database. Any other data access method would circumvent the security built into the RDBMS and potentially compromise data security and integrity.

Details presented in this chapter are largely transparent to end users of many ORACLE tools or applications. However, if you design applications, you may desire more control, especially if your application is large or complex. In particular, the programmatic interfaces offer more control at more levels than other ORACLE utilities. For more information you can refer to the precompiler manuals, such as the *Pro*FORTRAN User's Guide*.

What Constitutes a SQL Statement?

A SQL *statement* is a string of SQL text that is given to the RDBMS to execute. The text consists of SQL *reserved words*, words which have special meaning in SQL and are normally not used for any other purpose. For example, SELECT and UPDATE are reserved words and cannot be used as table names.

The statement must be the equivalent of a SQL "sentence," as in:

```
SELECT ENAME, DEPTNO FROM EMP
```

Only a SQL statement can be executed, whereas a "sentence fragment" such as:

```
SELECT ENAME
```

will simply generate an error indicating that more information is required. A SQL statement can be thought of as a very simple, but powerful, computer program or instruction.

Cursors and Context Areas

A SQL statement is associated with a *cursor* for processing. Cursors are essentially synonymous with *context areas*, which are discussed in Chapter 8. The distinction between a cursor and a context area is subtle.

A *cursor* is a work area, or a repository for information. It contains information summarizing the state of a SQL statement at any given point in time. Most importantly, a cursor is a named resource available to a user, and is a tool used for parsing. Each cursor is given a *cursor ID*

which is used to refer to it. A cursor can be in a number of different states or phases, depending on the action of the user.

A *context area* is the area of memory which actually contains the cursor's information. You might think of a cursor as a name for the context area.

A Simple Example of SQL Statement Execution

The following is a simplified look at what happens during execution of a SQL statement. This example uses a DML statement (UPDATE); queries (SELECTs) require additional steps. Details of processing all types of SQL statements appear later in this chapter.

Assume that you are using a Pro*FORTRAN program to increase the salary for all employees in a department. You might embed the following SQL statement in your program:

```
EXEC SQL UPDATE EMP SET SAL = 1.10 * SAL  
      WHERE DEPTNO = :DEPTNO
```

Assume that you are connected to ORACLE and have the appropriate privileges to update. `:DEPTNO` is a *host variable* containing a value for department number; a value for that variable must be substituted for `DEPTNO` by the host language program before the statement can be executed. When the SQL statement is executed, Pro*FORTRAN will copy the value of `DEPTNO` into the SQL statement.

The steps for executing this statement follow:

1. Create a cursor. A cursor is created, via a program interface call. The cursor is created independent of any SQL statement; it is created in expectation of any SQL statement. In precompiler programs, cursor creation may occur implicitly.

2. Parse the statement. The syntax of the SQL statement is scanned (parsed) and a representation of the SQL statement is loaded into the cursor. *Parsing* is the process of:

- translating a SQL statement
- loading it into a cursor
- checking authority to access referenced database objects
- determining the access path to be used to execute the statement
- identifying execution and resource requirements
- reserving those requirements.

For example, the definitions of database objects affected by the statement (such as tables or indexes) may be locked so that they do not change during execution of the statement.

3. Bind any variables. At this point ORACLE knows the meaning of the SQL statement but still does not have enough information to execute it. Specifically, it needs values for any variables in the statement; in this case it needs a value for :DEPTNO. This process is called *binding variables*. A program must specify either the actual value or the memory address in the program where the value can be found.

4. Execute the statement. At this point, the RDBMS has all necessary information and resources and so executes the statement.

Using a Cursor to Re-execute Statements

After each step or phase of execution, the cursor retains enough information to repeat the statement without starting over, as long as no other SQL statement has been associated with that cursor. In other words, once a statement has been parsed, you can repeat the bind phase using different values for the bind variables, and the statement need not be re-parsed. If the bind variables do not change datatype or value, you can also skip rebinding and simply re-execute. If variables are bound by reference, rebinding is unnecessary because ORACLE knows the addressing memory of the variables and can look for their new value automatically.

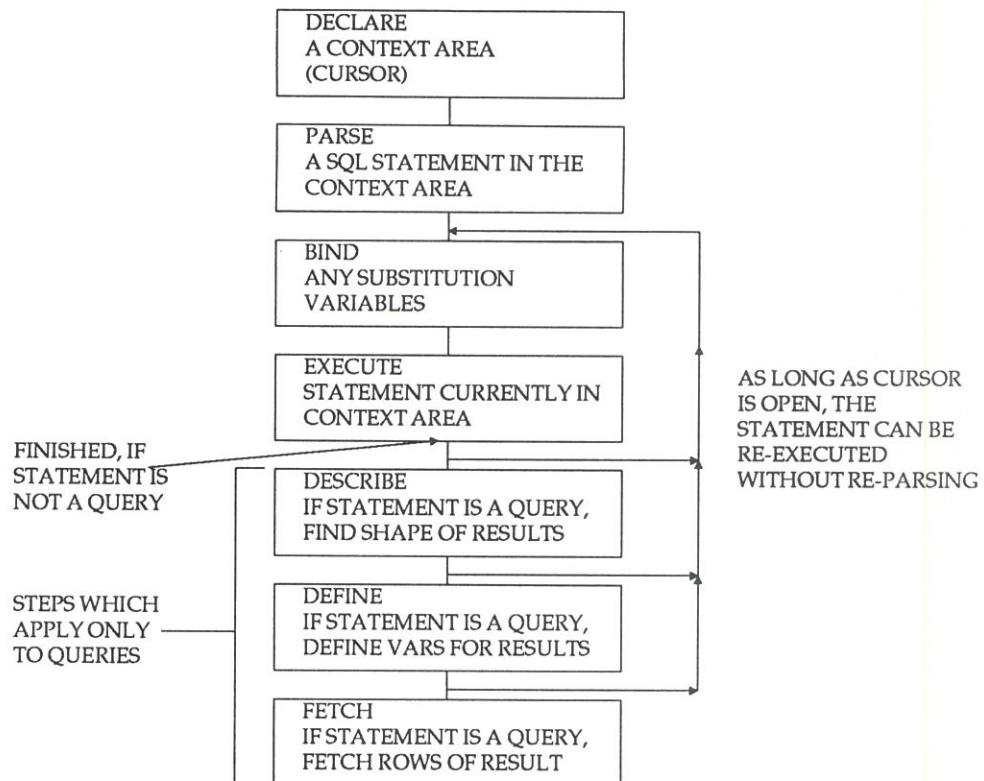
By opening several cursors, the parsed representation of several SQL statements can be saved. Repeated execution of several SQL statements can thus begin at the bind or execute step, saving the repeated cost of opening cursors and parsing.

Statement Processing in More Detail

As you read the information below, remember that for many ORACLE utilities all of this is done automatically for you. Most users need not be aware of this level of detail. However, designers of ORACLE applications may find this information useful when writing or fine tuning applications. For example, some ORACLE tools allow designers control over the number and reuse of cursors.

Figure 10-1 summarizes the stages of executing a SQL statement.

FIGURE 10-1
Stages of Processing a
SQL Statement



Opening and Closing Cursors

Although most ORACLE users rely on the automatic cursor handling of the ORACLE utilities, the programmatic interfaces offer application designers more control over cursors. For example, you can explicitly create a cursor via program interface calls to the RDBMS.

There is no absolute limit to the total number of cursors one user can have open at one time, subject to two constraints:

- Each cursor requires virtual memory, so a user's total number of cursors is limited by the memory available to that process.
- A system-wide limit of cursors per user is set by the parameter named `OPEN_CURSORS` found in the `INIT.ORA` file. Refer to Appendix D for details on `OPEN_CURSORS`.

Explicitly creating your own cursors can offer some advantages in tuning an application. For example, increasing the number of cursors can often reduce the frequency of parsing and improve performance. If

you know how many cursors may be required at a given time, you can make sure you can open that many simultaneously.

Refer to Chapter 19 for more information on tuning application performance.

Parsing Statements

During *parsing*, a SQL statement is passed from the user process to the RDBMS and prepared for execution. Many errors are caught at this phase of processing. During parsing the RDBMS:

- translates the statement, verifying it to be a valid SQL statement
- checks that the current user has adequate database privileges to perform the statement's actions
- performs data dictionary lookups to check table and column definitions
- resolves queries using synonyms and views into queries against the underlying tables
- for distributed queries, decomposes the query against tables on the various CPUs
- acquires parse locks on the objects (tables, indexes, views) required, so their definitions will not change during the statement's execution
- chooses the optimal access path to perform the actions (for example, choosing among various available indexes).

The parse locks are dropped when the cursor is closed (deallocated) or when another SQL statement replaces the first one (reused).

The parse phase is specifically designed to include processing requirements which can be done once no matter how many times the statement will be executed. ORACLE translates each SQL statement only once, re-executing that parsed statement during subsequent references to the statement.

Although the parsing of a SQL statement validates that statement, parsing only identifies errors which can be found **before statement execution**. Thus, certain errors will not be caught by parsing. For example, errors in data conversion or errors in data (such as duplicate values to be indexed by a unique index) and deadlocks are all errors or situations which can only be encountered and reported at execution.

Binding Variables

In the *bind phase* of statement processing, you specify data values to be used for the SQL statement's processing. End users of applications may be unaware that they are specifying bind variables because the ORACLE utility might simply prompt them for a new value.

To bind a variable, you must specify either a value (a constant) or the location of a value, so ORACLE can find the appropriate value at execution time. If you specify the location (binding by reference), you need not re-bind the variable before re-execution. You may change its value and ORACLE will look up the value on each execution, using the address.

Unless they are implied or defaulted, you may also specify a datatype and length for each value, in the event that ORACLE needs to perform datatype conversion.

Describing Results

The *describe phase* of execution is required for queries only. Queries are different from other types of SQL statements because they return data as results if they are successful. Whereas other statements return simply success or failure, a query may return several thousand rows. The results of a query are *always in tabular format*, and the rows of the result are fetched, either a row at a time or in groups.

The describe phase is not necessary unless the characteristics of the result are not known. For example, we know we want two columns for

```
SELECT ENAME, EMPNO FROM EMP
```

but we may not know what will result from:

```
SELECT * FROM DEPT
```

In this case, the describe phase is used to determine the characteristics (datatypes, lengths, and names) of a query's result.

Defining Output

Like the describe phase, the *define phase* of execution is used only for queries. In the define phase, you specify the location, size, and datatype of variables defined to receive each fetched value. ORACLE will perform datatype conversion if necessary.

Execution

During execution, the action requested by the SQL statement is performed. For some statements you can specify a number of executions to be performed. This is called *array processing*. Given n number of executions, then the bind and define locations are assumed to be the beginning of an array of size n . For more details on array

processing, see a programmatic interface guide such as the *Pro*C User's Guide*.

DDL Processing

The execution of DDL statements differs from the execution of DML statements and queries because the success of a DDL statement requires write access to the data dictionary. For DDL statements the parse phase actually includes the parsing, data dictionary lookup, and execution, all at the same time. There are no bind or define stages of a DDL statement.

Query Processing

Several issues relate particularly to query processing. Queries include not only explicit SELECT statements but also the implicit queries in other SQL statements. For example, each of the following statements require a query as a part of their execution:

```
INSERT INTO table SELECT ...
UPDATE table SET x = y WHERE ...
DELETE FROM table WHERE ...
CREATE table AS SELECT ...
```

In particular, queries:

- may require the describe and define phases
- are not actually executed in the execute phase. (Instead, preliminary work is done in preparation for the fetching of rows)
- require read consistency (see Chapter 12)
- may use temporary segments for intermediate processing (see Chapter 4).

Fetching Rows of a Query Result

In the case of queries, the execute phase is followed by *fetching* rows of the result. Rows are selected and ordered as requested by the query, and each successive fetch retrieves another row of the result until the last row has been fetched. Rows can be fetched into *arrays*, in which case the array size *n* determines the number of rows which are retrieved at each fetch.

11

TRANSACTION CONTROL

Success generally depends upon knowing how long it takes to succeed.
Charles Louis de Secondat Montesquieu

This chapter defines a transaction and describes the control that you have over when a transaction's changes become permanent. Topics include:

- the definition of a transaction (also called a logical unit of work)
- savepoints and read consistent "snapshots"
- the SQL statements used to control transactions:
 - COMMIT
 - ROLLBACK
 - SAVEPOINT
- the effect of issuing these statements explicitly or implicitly.

Application designers as well as DBAs will be interested in this material. Related information can be found in:

- Chapter 12 Concurrency and Locking
- Chapter 15 Database Recovery

Definition of a Transaction

A *transaction* (or a *logical unit of work*) is a sequence of SQL statements that ORACLE treats as a single entity. Most software applications consist of a number of types of transactions. A transaction can either be done repetitively, as in adding new rows of research data to a table of survey results, or somewhat randomly, as in "browsing the data" to become familiar with it or to ask some ad hoc questions.

A transaction can consist of one or many SQL statements (and thus can be short or long in duration). For example, all of the following are valid transactions:

- creating a table called TEMPTABLE
- inserting one new row into a table called VENDORS
- dropping an employee from one department, adding her to another department, changing her project assignments, and giving her a raise (regardless of how many SQL statements are used to do this or how many tables are actually updated).

In the last example, all changes to the appropriate tables can be treated as one transaction; only after all statements complete successfully are any made permanent.

A SQL statement that "executes successfully" is different from a "committed" transaction.

Executing successfully means that one statement was parsed and found to be a valid SQL construction, and that it executed without error. However, until the statement is committed as a part of a transaction, the issuer of the statement can undo the statement (undo all of its changes). A statement, rather than a transaction, executes successfully.

Being committed means that the issuer of the statement has either explicitly or implicitly said "make the changes in this transaction permanent." You commit a transaction, consisting of one or many SQL statements; you do not commit a statement. Until a transaction is committed, none of its changes are visible to other users. You can also choose to undo a transaction (roll it back). Both SQL statements COMMIT and ROLLBACK are discussed later in this chapter.

The Significance of Transactions

ORACLE ensures data consistency based on transactions rather than on single SQL statements (although a transaction can easily be just one SQL statement). For every transaction, either all of the changes made to the data are completed (made permanent in the database) or none of the changes are completed. If the operating system or a user program fails in the middle of a transaction, the database is automatically restored to the state it was in prior to the transaction starting.

The transaction model gives you more flexibility and control when working with data, and assures data consistency in the event of user process failure or system failure.

For example, if money is to be deducted from one account and added to another account, then both updates should either succeed together or fail together. If an error occurs making the updates, then neither update is made. The statements making up the transaction can be re-executed if necessary.

Starting and Ending a Transaction

A transaction begins when the first executable SQL statement is encountered. An *executable SQL statement* is a SQL statement which generates calls to the database, and includes all DML, DDL, and DCL statements.

A transaction ends when any of the following occurs:

- a COMMIT or ROLLBACK is invoked
- a DDL command is *issued* (such as CREATE, DROP, RENAME, ALTER) (The prior transaction is committed.)
- a DDL command *completes* (The transaction, consisting of just the DDL statement itself, is automatically committed.)
- user disconnects from ORACLE (such as EXEC SQL ... RELEASE) (The transaction is committed.)
- abnormal termination of a user process (The transaction is rolled back.)

After one transaction ends, the next executable SQL statement will automatically start the next transaction. If there is no final COMMIT or ROLLBACK work statement in a program, normal termination of the program (such as EXIT, as in SQL*Plus, or DISCONNECT) will commit the transaction or abnormal termination (such as a program ending before an EXIT or DISCONNECT from ORACLE) will roll it back.

If a single SQL statement fails, its effect is rolled back, but previous statements are not undone and may be committed or rolled back explicitly by the user. When a user's program fails in the middle of a transaction, ORACLE detects the error and restores the data to its state

prior to the transaction. If the operating system fails (rather than the user process), then the data is restored when the system is restarted, as a part of instance recovery.

Read-Only Transactions

Read only transactions are useful when querying or reporting on data in several database tables. For example, you might often want data in a group of related tables to be consistent while you execute several queries against it.

Using the SQL statement `SET TRANSACTION READ ONLY` you can indicate that a particular transaction will be in read mode only. For more information on this statement, refer to Chapter 12.

How are Transactions Defined in ORACLE Utilities?

Usually the ORACLE tool or application you use controls what constitutes a transaction and your primary concern is whether to commit or roll back a given transaction.

In general, only applications designers using the programming interfaces to ORACLE are concerned with which types of actions should be grouped together as one transaction. However, they should be specifically concerned with defining transactions properly so work is accomplished in logical units of work.

A transaction should consist of all of the necessary parts for one logical unit of work — no more and no less. Data in all referenced tables should be in a consistent state before the transaction begins and after it ends. Transactions should consist of only the SQL statements that comprise one consistent change to the data.

For example, a transfer of funds (say \$1000) between two accounts should include the debit to one account of \$1000 and the credit to another account of \$1000. Both actions should either fail or succeed together. The credit should not be committed without the debit.

Or, if two accounts are debited rather than one (two debits of \$250 and \$750), then all three actions (two debits and one credit) should be committed simultaneously. Another non-related action, such as a new deposit to one account, should not be included in the transfer of funds transaction.

Controlling Transactions with SQL Statements

Three SQL statements are used to control when transactions take effect or are rolled back:

```
COMMIT WORK  
SAVEPOINT  
ROLLBACK WORK
```

The statements COMMIT and ROLLBACK finish one unit of work and begin another. COMMIT makes the changes "permanent" and ROLLBACK "undoes" all changes made in that transaction. SAVEPOINT is similar to taking a "snapshot" of the data, so that later you can either save your changes or undo them as far back as the savepoint.

The COMMIT WORK Statement

The COMMIT WORK statement performs the following tasks:

- makes "permanent" all changes performed in the current transaction (naturally, the data may be changed by future updates)
- erases all savepoints in that transaction
- ends the transaction
- releases the transaction's locks.

The syntax for COMMIT is:

```
COMMIT [WORK]
```

The keyword WORK is optional and makes no difference.

The recommended practice is to explicitly end transactions in application programs using the COMMIT WORK (or ROLLBACK WORK) statement, including the last transaction before disconnecting from ORACLE. If you do not explicitly commit the transaction and the program terminates abnormally, the last uncommitted transaction will be rolled back.

What Happens When You Commit?

Before you commit a transaction in which you have altered data, the following have occurred:

- The RDBMS has generated rollback records in memory.
- The RDBMS has generated redo log records in memory.
(These changes may go to disk before a transaction is committed.)
- The changes have been made to the database buffers.

When the transaction is committed, explicitly or implicitly, the following occurs:

- The transaction is marked "complete."
- The redo log entry is written to disk (the online log files) if it hasn't been already.
- Locks held on rows and tables are released but parse locks held by an open cursor are not.

Subsequent to (but not necessarily at the same time as) the commit, the database buffers are written to the database. Rollback information need never go to disk for a committed transaction.

The writing of the redo log entry suffices as a permanent record of the transaction, because the database can always be reconstructed from information in the log files.

Using SAVEPOINTS to Save Work

Savepoints can be used to sub-divide a transaction into smaller parts. You can arbitrarily save your work at any point in time, with the option of later committing that work or rolling it back. Thus, for a long transaction, you can save portions of it as you proceed, finally committing or rolling back.

The syntax of the SQL statement for this purpose follows:

```
SAVEPOINT savepoint_id
```

where *savepoint_id* is a SQL identifier (and must follow the same rules as for other database objects) or a host variable, as in:

```
SAVEPOINT add_empl  
SAVEPOINT insertsmade
```

If you create a second savepoint with the same identifier as an earlier savepoint, the earlier savepoint is erased. After a savepoint has been created, you can either COMMIT the work

COMMIT WORK

or rollback to the savepoint:

ROLLBACK WORK TO savepoint_id

By default, the maximum number of active savepoints per process is 5. An *active savepoint* is one which has been specified since the last commit or rollback. This limit can be changed by changing the value for parameter SAVEPOINTS in the INIT.ORA file. The absolute limit is 255 active savepoints.

Figure 11-1 shows an example of using a savepoint during an interactive transaction to undo a portion of a transaction.

```
UPDATE EMP SET DEPT = 'US SALES' WHERE DEPT = 'SALES'  
SAVEPOINT USDONE
```

```
UPDATE EMP SET DEPT = 'INT'L SALES' WHERE MGR = 6745  
SAVEPOINT ILDONE
```

```
UPDATE EMP SET MGR = 6745 WHERE MGR = 5902  
ROLLBACK TO USDONE
```

```
UPDATE EMP SET MGR = 6745 WHERE MGR = 5902  
UPDATE EMP SET DEPT = 'INT' SALES' WHERE MGR = 6745  
COMMIT
```

Savepoints are useful in interactive programs, because you can create and name intermediate steps of a procedure. This allows you more control over longer, more complex procedures. For example, you can use savepoints throughout a long complex series of updates, so that if you make an error, you need not resubmit every statement.

Savepoints are useful in application programs in a similar way. If a procedure contains several functions, a savepoint can be created before each function begins. Then, if a function fails, it is easy to return the data to its state before the function began and then re-execute the function with revised parameters or perform a recovery action.

FIGURE 11-1
**Example of Savepoints,
Commits, Rollbacks**

Statement Level Rollback

If a SQL statement causes an error, such as attempting to insert a duplicate value in a unique index, that statement is rolled back. The effect of the rollback is as if that statement were never executed and as though the system performed a savepoint immediately before attempting that statement.

Errors which cause rollbacks are errors which are discovered during execution, such as attempting to insert a duplicate value in an index or an invalid number into a numeric column. Syntax errors in a SQL statement are discovered at parse time and do not cause rollbacks. Single SQL statements may also be rolled back in order to resolve deadlocks (and a warning appears alerting the user to this situation).

Thus, a SQL statement which fails causes the loss only of any work it would have performed itself; *it does not cause the loss of any work which preceded it in the current transaction*. If the statement is a DDL statement, the commit which immediately preceded it is not undone.

This feature is known as *statement level rollback*. ORACLE performs statement level rollback by creating an implicit savepoint before executing any SQL statement. You cannot refer to this savepoint directly.

The ROLLBACK WORK Statement

The ROLLBACK statement is used to undo work. The syntax is:

```
ROLLBACK [ WORK ]      [ TO [ SAVEPOINT ] savepoint_id ]
```

The keyword WORK is optional and makes no difference.

Using ROLLBACK without a TO SAVEPOINT clause:

- ends the transaction
- undoes all changes in the current transaction
- erases all savepoints in that transaction
- releases the transaction's locks.

Using ROLLBACK with a TO SAVEPOINT clause:

- rolls back just a portion of the transaction
- retains the savepoint rolled back to, but loses all subsequent ones. In Figure 11-1, the ROLLBACK loses the work saved in savepoint ILDONE, but retains the savepoint USDONE.
- releases all table and row locks acquired since the savepoint was taken but does not release the transaction (TX) lock.

The transaction lock is released on a commit or rollback. If users are waiting for rows that your transaction had locked, they may wait until your TX lock is released.

Recommended practice is to explicitly end transactions in application programs using either COMMIT WORK or ROLLBACK WORK. If you do not explicitly commit the transaction and the program terminates abnormally, the last uncommitted transaction is rolled back.

Invoking COMMIT or ROLLBACK

COMMIT and ROLLBACK can be invoked either explicitly by a program or user, or implicitly (automatically) by ORACLE.

Explicit Commits/Rollbacks

In most ORACLE products, you must explicitly commit (or roll back) your transactions using either COMMIT or ROLLBACK WORK. You can also use the SQL statement SAVEPOINT in these products.

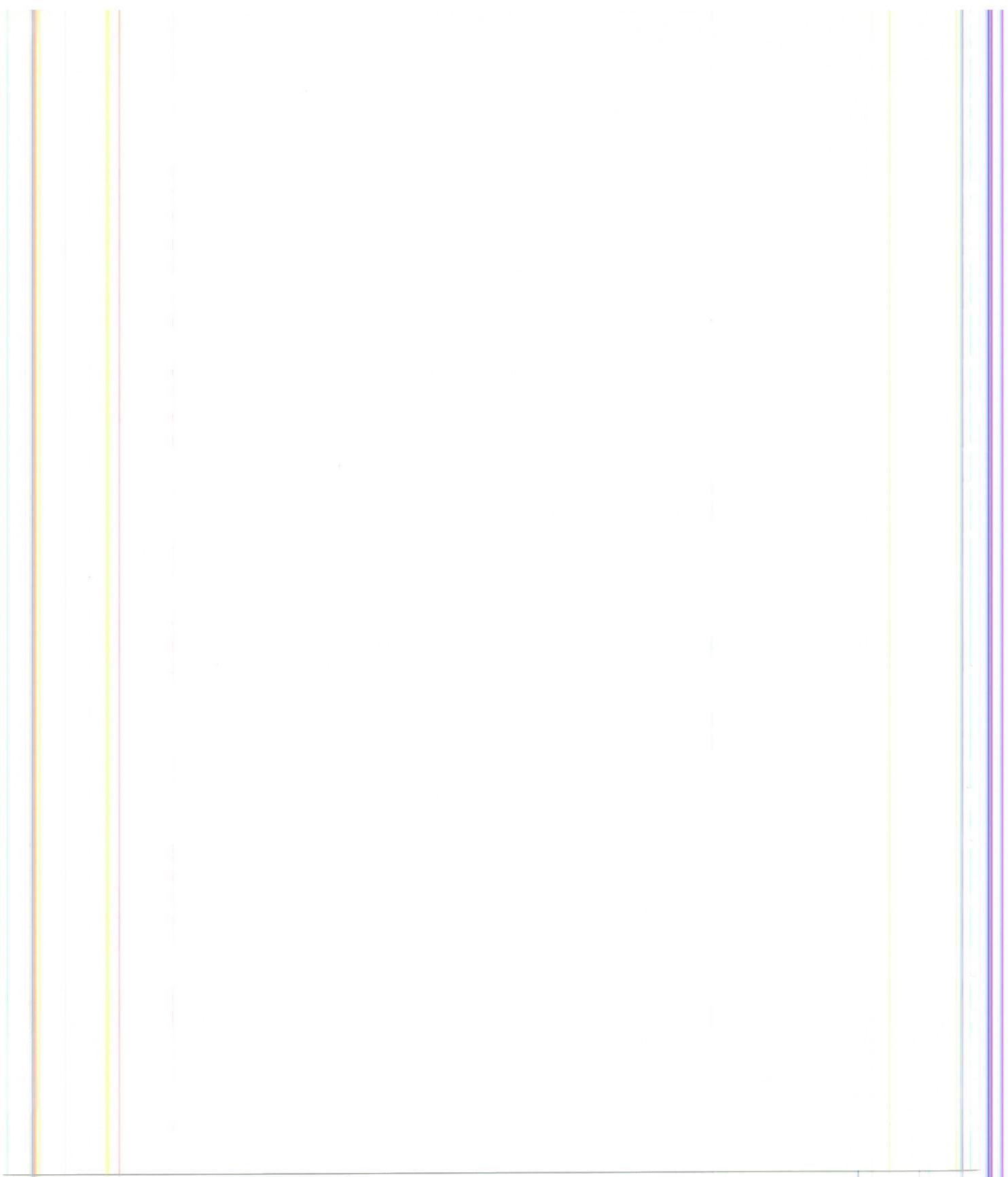
Implicit (Automatic) Commits/Rollbacks

Implicit commits occur in the following circumstances:

- before a DDL statement
- after a DDL statement
- at normal disconnect from the database.

Data definition statements (DDL statements — such as CREATE, DROP, and GRANT) always cause commits at the time they are executed. If you enter a DDL statement having just entered several DML statements, the DDL statement causes a commit to occur prior to its own execution, ending the current unit of work. Then, if the DDL statement executes successfully, it too is committed.

Implicit rollbacks occur if a process or an instance terminates abnormally. Implicit statement level rollbacks occur on statement execution error.



12

CONSISTENCY AND CONCURRENCY

Make 'em laugh; make 'em cry; make 'em wait.

Charles Reade

Thou canst not say I did it; never shake

Thy gory locks at me.

Shakespeare: *Macbeth* III.iv

This chapter discusses issues dealing with concurrency (when multiple users access the same data in a database) and data integrity. For example, what happens when two users attempt to simultaneously update the same row?

Topics covered in this chapter include:

- read consistency
- read-only transactions
- default locking behavior
- overriding default locking
- using SQL*DBA to monitor current locks and latches
- deadlock detection
- the INIT.ORA parameters SERIALIZABLE and ROW_LOCKING.

Portions of this chapter directly address the differences between ORACLE RDBMS Version 6 with and without the transaction processing option.

The contents of this chapter are more technically complex than will be useful to many database users. The contents will be of interest primarily to designers of applications accessed by many simultaneous users and to DBAs who are tuning performance. These readers may also find the following sections relevant:

- Appendix B, particularly the SQL*DBA reference for using MONITOR LOCKS
- Chapter 11 Transaction Control

General Concurrency Issues

A primary concern of any database management system is how to control *concurrency*, or the accessing of the same data by many users. Without adequate concurrency controls, data may be updated or changed improperly or out of sequence, compromising data integrity.

Concurrency concerns apply both to *data* and to *structures*. Changes to data should be guaranteed to be made in the order the changes actually occur. Similarly, changes to the definitions of structures, such as tables or indexes, should also occur in the proper order. If one person is accessing a table, its contents or structure should not appear to change until that person is done viewing the table.

Data or data structures might become compromised in the following ways:

- "lost updates" (one update overwriting another)
- two concurrent updates resulting in two identical index values in an index declared unique
- updates to a table simultaneous with another user deleting a table or altering its definition.

Obviously, if many people are accessing the same data in the same table, one way of guaranteeing integrity in the data and the table structure is to make each person wait his turn.

The goal of the ORACLE RDBMS is to reduce that wait so it is either non-existent or negligible to each user. All SQL statements (queries, DML, or DDL) should proceed with as little interference as possible, and destructive interactions between transactions must be prohibited. Neither performance nor data integrity can be compromised.

ORACLE resolves issues such as concurrent updates to the same data using various types of *locks* and *modes of operation*. These features are based on the concept of a *transaction*. It is the application designer's responsibility to assure that transactions are defined properly, as described in Chapter 11.

There are many types of locks, some of which can be controlled by users and some which cannot. You can see the current locks in a running ORACLE system using the SQL*DBA command MONITOR. See Appendix B for the syntax and description of the command MONITOR and for examples of its displays.

Read Consistency

Read consistency is the consistency model supported by the ORACLE RDBMS. Read consistency:

- guarantees that the data seen by a statement (such as a query) will not change during statement execution
- assures that readers of database data do not wait for writers of the same database blocks.

The simplest way to think of read consistency is to imagine that a "snapshot" of a table's data is taken whenever a query against that table is executed. ORACLE's read consistency model is also called a "multi-version" consistency model because multiple versions of the same table may appear to exist simultaneously.

The ORACLE RDBMS must create a read consistent set of data when the table is being simultaneously queried and updated. When a read consistent view is necessary, the RDBMS uses information in the rollback segments to generate a read consistent view of a table's data for a query. No read locks are ever required by ORACLE queries. This ensures that readers do not interfere with writers and that writers do not interfere with readers.

The "snapshot" is taken during the execute phase. Though you can imagine it as a snapshot, it is not actually a full copy of the data. Instead, when other users have changed the queried data since the query began, ORACLE reconstructs the older data blocks that it needs, using rollback segments. Only blocks which have changed since the query began need to be reconstructed. Read consistency is thus assured and the performance impact is acceptable.

The read consistency model is significant primarily to applications designers. For example, it may influence your coding if you know that

the EXEC SQL OPEN command in precompiled programs will start execution and set up the "snapshot," and that any changes made to data after the snapshot is taken will not be seen by the application. You may wish to put the call later in a program, rather than earlier.

Read Consistency Examples

For example, if BARB queries table EMP, the results which are returned (fetched) for her query are the data that existed in EMP at the time her query started. These results include all DML changes made by BARB prior to executing her query (whether committed or uncommitted) and all committed changes to table EMP (made by any user) as of that time. However, changes that BARB makes to EMP after executing her query do not appear in the query's results, nor do any changes made to EMP by other users that were uncommitted when her query began.

Consider two users, Bert and Ernie, both accessing a general ledger. Bert queries the database (call that TX-B, for "transaction-B"), totalling DEBITS and CREDITS, and expects their totals to match. At the same time Ernie might enter a transaction (TX-E) by debiting ACCT1 and crediting ACCT2.

If Bert queries while Ernie is in mid-transaction, read consistency ensures that Bert gets the same balance for DEBITS and CREDITS. If the application is designed properly, Ernie's transaction will be committed only when his debit entries equal his credit entries. Then Bert can see only the results of complete transactions; in this case, his query (TX-B) will see a total for DEBITS and CREDITS before TX-E occurs. Or, if Bert's query began after Ernie committed TX-E, Bert would see the new totals, but they would still be equal.

If Ernie updates while Bert is in query, Bert will see only the pre-update values, even if Ernie commits the new values before Bert reads the update rows.

The "Snapshot Too Old" Message

Occasionally the RDBMS may not be able to return a read consistent set of results. This may happen when much update activity occurs (on the table or in the rollback segment), because not enough information remains in the rollback segment to reconstruct the older data. In this event, error 1555 will result:

```
ORA 1555 snapshot too old (rollback segment too small)
```

You can avoid this situation by obtaining a shared lock on the table you are querying, thus prohibiting any other exclusive locks during the transaction. If the situation arises frequently, you can also use more or larger rollback segments.

Introduction to Locking

The ORACLE RDBMS uses locks to control concurrent access to data. *Locks* are mechanisms intended to prevent destructive interaction between users accessing ORACLE data. *Destructive interaction* can be interpreted as any interaction which incorrectly updates data or incorrectly alters underlying data structures (such as table or column definitions, indexes, or user grants).

Locks are used to achieve two important database goals:

Consistency Assure users that the data they are viewing or changing is not changed until they are through with it.

Integrity Assure that the database's data and structures reflect all changes made to them in the correct sequence.

Thus, to summarize, locks are used to insure data integrity while allowing maximum concurrent access to the data by unlimited users.

ORACLE locking is fully automatic and requires no user action. Implicit locking occurs for all SQL statements, depending on the action requested. You need never explicitly lock any resource; ORACLE's default locking mechanisms will protect the data.

However, under some circumstances, you may want to override default locking in order to tune or optimize an application. This is most likely to occur when writing applications using the programmatic interface products.

What Resources can be Locked?

A lock can be thought of as assigning a database user "temporary ownership" of a database *resource*, such as a table or a row of data in a table. Resources include two general types of objects:

- *user objects* such as tables and rows (structures and data)
- *system objects* not visible to users, such as shared data structures in the SGA and database buffers.

A lock on a resource is *acquired* by a user when the user wants to keep another user from doing something also requiring that resource. The lock is *released* when certain events occur and the first user no longer requires the resource.

Throughout its operation the ORACLE RDBMS automatically locks numerous structures on behalf of users. Users can explicitly lock *rows* of

a table using the SELECT ... FOR UPDATE statement or entire *tables* using the LOCK TABLE statement.

Data dictionary tables cannot be locked explicitly. Individual data dictionary tables are often locked, however, by the ORACLE RDBMS in the course of normal database operation. These locks are of very short duration; see "Dictionary Locks" later in this chapter for more information.

Duration of Locks

The duration of a lock varies, depending on what it is protecting. Some locks (internal latches) are held for very short periods, during the manipulation of a resource. Other locks are held longer (for example, table and row locks are held for the length of a transaction). Still other locks are held the entire time the database is open (for example, the locks that guarantee that only one instance writes to a given rollback segment).

All data locks are released on a commit. Table locks are released by rolling back to a savepoint. Row locks are not released by rolling back to a savepoint.

Who can Explicitly Lock?

Applications designers using the SQL locking statements in their programs must make sure that the ORACLE users requesting locks have sufficient database privileges to obtain the locks, using the following guidelines:

- DBAs can lock any table.
- Other users can lock tables they own or any table on which they have any table privilege (such as SELECT, INSERT, UPDATE, DELETE).

Default vs. Manual Locking

Most locks either cannot or need not be altered. However, you can request specific *data locks* on tables or rows, using SQL statements, when it is to your advantage to override default locking.

Following sections of this chapter describe default locking behavior. Later sections describe how to override default locking and the SQL statements LOCK TABLE and SELECT FOR UPDATE used for this purpose. Also described is the SET TRANSACTION READ ONLY command, which alters ORACLE's behavior for multiple queries in a transaction that does no updates.

Types of Locks

Locks in ORACLE fall into the following general categories:

Data locks (also called DML locks) These lock and protect data. Table locks lock an entire table, while row locks lock just selected rows.

Dictionary locks These locks protect the structure of database objects. For example, they protect the definitions of tables or views. Dictionary locks are of two types: *parse locks* and *DDL locks*.

Internal locks and latches These lock internal database structures and are entirely automatic.

To view current locks, use the SQL*DBA commands MONITOR LOCK or MONITOR LATCHES. A summary of all types of locks appears in Figure 12-1.

FIGURE 12-1
Summary of All
Database Locks

Lock Type	Appearance
DATA LOCKS (or DML LOCKS)	
Table Locks	TM
EXCLUSIVE	X
SHARE UPDATE	RS
SHARE ROW EXCLUSIVE	SRX
ROW SHARE	RS
ROW EXCLUSIVE	RX
SHARE	S
Row Locks (Transaction Lock)	TX
DICTIONARY LOCKS	
Parse Locks	TD (S)
DDL Locks	TD (X)
Temporary Segment	TT (S or X)
INTERNAL LOCKS	
LATCHES	
Dictionary Cache Locks	numerous
Shared	numerous
Exclusive	
File and Log Management Locks	
Tablespace and Rollback Segment Locks	

Data or DML Locks

The purpose of *data locks* or *DML locks* is to guarantee data consistency when data is being accessed concurrently by multiple users. Data locks prevent interference of simultaneous DML operations, thus the name DML locks. For example, a row in a table should only be updated by one transaction at one time, though it may be updated rapidly by many transactions in succession.

Data Locking Behavior for ORACLE with the Transaction Processing Option

Note: The following sections describe default locking behavior for ORACLE with the transaction processing option. These sections describe how ORACLE locks tables and rows for certain SQL statements when you make no locking specifications.

Thus, these sections presume that two INIT.ORA parameters are set to the default for ORACLE with the transaction processing option, as follows:

```
SERIALIZABLE = FALSE  
ROW_LOCKING = ALWAYS
```

(See the end of this chapter for the effects of setting these parameters to TRUE in various combinations.)

Later sections describe situations when you might wish to alter the default locking and how you can do so.

Read Consistent Statements for Queries, Updates, and Deletes

The SQL statements SELECT, INSERT ... SELECT, UPDATE, and DELETE all query data, either explicitly or implicitly. Each of these statements uses a query to determine which data will be affected (selected, inserted, updated, or deleted, respectively).

A SELECT statement is an explicit query and may have nested queries. An INSERT statement can use nested queries. UPDATE and DELETE statements must use WHERE clauses in order to affect only some rows in a table rather than all rows and may use subqueries.

A *read consistent result is guaranteed for every query*, including implicit queries, guaranteeing data consistency and requiring no action on the user's part.

However, if you want to read data and change it later *in the same transaction*, then you may wish to use manual locking, as described in "Overriding Default Locking."

Default Locking for Queries

Queries, as they only read data and do not alter it, are the SQL statements least likely to interfere with other SQL statements. However, users executing queries are concerned that the data they see is in a consistent state.

The following characteristics are true of queries that do not use the FOR UPDATE clause:

- A query requires no data locks.
- A query never waits for any data locks to be released; it can always proceed.
- A query always returns a read consistent view of the data.
- Other transactions can access the table being queried, including the particular rows being queried.

A transaction executing a query never has to wait for data resources, even when the underlying data is being accessed or altered by other transactions. No data locks are required on rows or tables.

The ORACLE RDBMS always establishes a read consistent snapshot of the data at the time the query is executed. The query sees all data that was committed as of the start of the query, plus any updates that are made in the transaction of which the query is a part.

Figure 12-2 summarizes the default locks obtained by SQL statements under ORACLE with the transaction processing option.

FIGURE 12-2
Default Locking with the
transaction processing option

<i>SQL Statement</i>	<i>Row Lock</i>	<i>Table Lock</i>
SELECT	-- no lock obtained --	
SELECT FOR UPDATE	X	RS
LOCK table IN mode MODE:		
EXCLUSIVE	--	X
SHARE UPDATE	--	RS
SHARE ROW EXCLUSIVE	--	SRX
ROW SHARE	--	RS
ROW EXCLUSIVE	--	RX
SHARE	--	S
INSERT	X	RX
UPDATE	X	RX
DELETE	X	RX
DDL/DCL	--	X

Default Locking for Updates and Deletes

The read consistency model also protects the integrity of results returned by DELETEs and UPDATEs. These statements also use consistent snapshots for their implicit queries.

However, in a busy system, other users may have just committed new data in the rows you wish to update or delete. If these changes occurred *after* your statement started but *before* your change was made, the ORACLE read consistency model requires extra work. In environments supporting high concurrent update activity on the same table, a DELETE or UPDATE statement that affects a high number of rows may take disproportionately longer than the same statement when it is executed on very few rows or when there is less concurrent activity on that table.

The following are characteristics of UPDATEs, DELETEs, and INSERTs:

- A query on behalf of an UPDATE, DELETE, or INSERT will always begin with a read consistent view of the data.
- The statements acquire exclusive locks (X) on the *rows* to be modified and row exclusive locks (RX) on the *tables* containing those rows.
- Other transactions can query the rows to be modified, but cannot alter them until the current transaction has released the row locks.
- Other transactions can do any action on other rows in the table, even in the same block as rows affected and even while the locks are held.
- The row and table locks are released when the transaction is committed.
- At a rollback to a savepoint, locks acquired after the savepoint are released.

Overriding Default Locking

Manual locking may be preferable to default locking when:

- you want to see multiple tables in a consistent state (data across all tables is consistent). An example would be one master and several detail tables. You can achieve this only through non-default locking or read only transactions.
- you want no one else to be able to change data you have looked at until you have committed your transaction, regardless of whether or not you change the data.
- you don't want your statement to wait for any other transaction to complete. If you request an exclusive lock then you will only wait for the exclusive lock, if you wait at all.
- you want to perform a "repeatable read" (you want to query the same table repeatedly in one transaction knowing no data has changed). You can achieve this only through non-default locking or read only transactions.

You can start read only transactions with the SQL statement SET TRANSACTION READ ONLY and you can override default locking by using two SQL statements: the LOCK TABLE statement and the SELECT FOR UPDATE statement. These statements are used to acquire data locks only; you cannot directly control dictionary locks, internal locks, or latches.

Using the LOCK TABLE Statement

The LOCK TABLE statement manually overrides default locking by creating a data lock in the mode specified. The syntax of the LOCK statement is:

```
LOCK TABLE table [,table] ... IN
  { ROW SHARE |
    ROW EXCLUSIVE |
    SHARE UPDATE |
    SHARE |
    SHARE ROW EXCLUSIVE |
    EXCLUSIVE }
  MODE [ NOWAIT ]
```

where:

<i>table</i>	indicates the names of tables, views, or synonyms to be locked. In the case of views, the locks are actually placed on the underlying tables. Data dictionary tables cannot be locked.
lock modes	indicates one of six lock modes; the modes are described below. (ROW SHARE and SHARE UPDATE are identical so there are effectively five types of lock modes.)
NOWAIT	Indicates that you do not wish to wait to acquire a lock. If the resource is unavailable, control is returned immediately to your application and you may continue with other work or simply wait before retrying to acquire the lock. Default locking (if you omit NOWAIT) will wait for a lock if the resource is currently unavailable. The wait has no set limit.

Shared and Exclusive Locks

Lock modes range in restrictivity from several types of *shared* locks to *exclusive* locks.

share locks	Various share locks indicate that the locked resource can be shared in different ways. That is, multiple users can hold various share locks on one single resource.
exclusive lock	An exclusive lock is the most restrictive type of lock. It prohibits all sharing of the resource (although querying data is never prohibited). That is, the first user to exclusively lock a resource has the sole ability to alter the resource until he releases the lock.

Descriptions of Lock Modes

Data locks can be obtained in one of several modes. The *mode* of a lock determines what other locks on the same resource (table or row) can exist simultaneously, thereby determining what other actions are compatible with that lock. For example, while one user has an exclusive lock on a table no other user can update rows in that table.

If you find the names of the locks confusing, you may find it helpful to concentrate on the interactions of the various locks rather than their names.

Exclusive (X)	Exclusive locks allow queries on the locked resource but prohibit any other activity on the resource.
Share (S)	Share locks allow queries but prohibit updates to a table.
Row Share and Share Update (RS)	Row share locks allow concurrent access to a table. They prohibit other users from locking the entire table for exclusive access. Row share is synonymous with share update (the latter mode is included for compatibility with earlier versions of the ORACLE RDBMS).
Row Exclusive (RX)	Row exclusive locks are the same as row share locks, but also prohibit locking in share mode. These locks are obtained when updating, inserting, or deleting.
Share Row Exclusive (SRX)	Share row exclusive locks are used to look at a whole table to do selective updates and to allow other users to look at rows in the table but not to lock the table in share mode or to update rows.

Though these modes of locks apply primarily to data locks, note that some dictionary locks and internal locks borrow some of the mode terminology.

Figure 12-3 shows the data lock modes and the modes with which they can coexist.

FIGURE 12-3
Data Lock Mode Compatibilities

Lock Mode Abbreviation	Actions Prohibited	Actions Allowed	SQL Statement Obtained by
RS (and share update)	X exclusive access:	updates: RS, RX, S, SRX	SELECT..FOR UPDATE
RX	exclusive access for reading/writing: S, SRX, X	updates: RX, RS	UPDATE, INSERT, DELETE
S	updates: RX, SRX, X	locking rows: RS, S	LOCK TABLE table IN SHARE MODE
SRX	reading whole table: RX, S, SRX, X	locking rows: RS	LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE
X	everything except queries: RS, RX, S, SRX, X	queries	LOCK TABLE table IN EXCLUSIVE MODE and most DDL statements

Lock Escalation

In some database systems, *lock escalation* occurs when numerous locks are held at one level (such as row locks) and the RDBMS automatically changes those locks to a different lock at a higher level (such as a table lock). Thus, the number of locks has been reduced, but the granularity of what is being locked has increased. Lock escalation greatly increases the likelihood of deadlock. (Deadlocks are described later in this chapter.) The ORACLE RDBMS does not escalate locks.

Lock Conversion

Lock conversion occurs when a lock in one mode is automatically converted to a lock in another mode. Lock conversions may occur, for example, if a user has one lock on a table and requests a different lock. The resulting lock may be different from the one requested, depending on how the ORACLE RDBMS performs the conversion.

When to Use SELECT FOR UPDATE

In addition to using the LOCK table statement, you can override default locking with the SELECT FOR UPDATE statement. SELECT FOR UPDATE is used in anticipation of performing updates and acquiring exclusive row locks (as the UPDATE statement does).

The difference between a SELECT FOR UPDATE and UPDATE is that the SELECT FOR UPDATE locks the rows earlier. First it executes the query to identify the rows that will be updated or deleted, and then it locks the set of rows. Optimistic locking is used as explained earlier.

SELECT FOR UPDATE is recommended when you need to lock a row without actually changing it. For example, if you intend to base an update on the existing values in a row, you need to make sure the row is not changed by someone else before your update.

A SELECT FOR UPDATE query terminates (and its locks are released) at COMMIT or ROLLBACK. It is better to use SELECT FOR UPDATE for a few rows than for a large number of rows.

Using SET TRANSACTION READ ONLY

By default, the consistency model for the ORACLE RDBMS guarantees that the results of one statement are consistent. However, in some situations, you may want to run many statements against a set of data you know is consistent. In these cases you would prefer that the consistency model apply to a transaction (multiple statements) rather than to a single statement.

Similarly, if you want to run queries against multiple tables and you will be doing no updating, you may prefer a read only transaction. After indicating that your transaction will be read only, you can execute as many queries as you like against any database table, knowing that your results are consistent with each other. That is, the results from any table are consistent in time with the results from any other table.

The SQL statement SET TRANSACTION READ ONLY is used to begin a read only transaction. The following guidelines pertain to this SQL statement:

- The SET TRANSACTION READ ONLY statement must be the first statement in the transaction, or an error is returned.
- Only queries (or a statement to terminate the transaction) are allowed in the transaction, or an error is returned (for example, a SELECT FOR UPDATE will receive an error).
- A COMMIT, ROLLBACK, or DDL statement will terminate the read only transaction (as DDL statements cause implicit commits).
- During the read only transaction, all queries refer to the same snapshot of the database, providing a multi-table, multi-query read consistent view.
- Other users may continue to query or update data as usual.
- The read only transaction will see only data that was committed when it began; furthermore, because no updates can be made in the transaction, it continues to see that data in that state until the transaction is terminated.
- Long running read only transactions may receive an error to the effect that the "snapshot is too old," in which case the transaction should be terminated and a new transaction started, if desired.

Dictionary Locks

A dictionary lock is acquired automatically by ORACLE RDBMS on behalf of any user requiring it. Users cannot directly request dictionary locks. While individual data dictionary objects can be locked, the data dictionary is never globally locked.

The purpose of a *dictionary lock* is to protect the definition of a database object (such as a table, index, or cluster) while that object is being used (that is, while it is referenced in a currently parsed statement).

Parse Locks

A dictionary lock held in share mode is called a *parse lock*. It locks an object, beginning with the parse of a SQL statement referencing the object, and ending when the cursor is closed or when another statement is parsed in the cursor.

DDL Locks

A dictionary lock held in exclusive mode is called a *DDL lock*. It is obtained for a SQL statement which is potentially incompatible with other statements referencing the same object. All DDL statements (such as CREATE, ALTER, DROP) require the exclusive lock *except*:

```
AUDIT and NO AUDIT  
GRANT ...  
COMMENT ...  
CREATE SYNONYM  
CREATE VIEW  
CREATE TABLE (if the table is not clustered)  
REVOKE ...
```

These DDL statements do not require exclusive dictionary locks and thus allow parsed cursors on the objects they reference.

Dictionary Locks and Clusters

Parse locks apply to entire clusters if one table in a cluster is locked. When multiple tables are clustered, their contents share common physical data blocks. Although this does not affect the behavior of row level locks (data locks), a DDL lock placed on one table in a cluster is placed equally on all tables in that cluster. Thus, if a statement is open on a table in a cluster, then a table cannot be created in that cluster until the statement finishes. This should be considered when designing data schema and applications for clustered data.

Internal Locks and Latches

Internal locks and latches protect internal database and memory structures (these structures are inaccessible to users). Users have no need for control over their occurrence or duration. The following information will help you interpret the displays resulting from the SQL*DBA commands MONITOR LOCKS or MONITOR LATCHES.

Latches

Latches are simple, low level serialization mechanisms to protect shared data structures in the SGA. For example, they protect the table of users currently accessing the database and the table describing the blocks in the buffer cache. A process acquires a latch for a very short time while manipulating or looking at one of these structures. The implementation of latches is very operating system dependent, particularly in regard to whether a process will wait for a latch and how long.

Several different latches appear in the MONITOR LATCHES display; see Appendix B for more details.

Internal Locks

Internal locks are higher level, more complex mechanisms than latches, and serve a variety of purposes. Details on the three categories follow.

Dictionary Cache Locks

These locks are of very short duration, and are held on entries in dictionary caches while the entries are being modified or used. They insure that users parsing statements do not see inconsistent definitions and can be held shared or exclusive.

Shared locks are released when the parse is complete. Exclusive locks are released when the DDL operation is complete.

File and Log Management Locks

These locks protect different files. For example, one lock protects the control file so that only one process can change the file at a time. Another lock coordinates the use and archiving of the redo log files. Database files are locked to ensure that multiple instances mount a database in shared mode or one instance mounts it in exclusive mode.

These locks are of particular importance in shared disk systems. Because they indicate the status of files, these locks are necessarily held for a long time.

Tablespace and Rollback Segment Locks

These locks protect tablespaces and rollback segments. For example, all instances accessing a database must agree on whether a tablespace is online or offline. Rollback segments are locked so that only one instance can write to a segment.

Monitoring Locks with SQL*DBA

Whenever a database is running, the DBA can view current locks or latches for an instance by using SQL*DBA.

1. Run SQL*DBA.
2. CONNECT to the database with a DBA username, as in:

```
SQLDBA> CONNECT SYSTEM/MANAGER
```

3. Enter either:

```
SQLDBA> MONITOR LOCKS  
SQLDBA> MONITOR LATCHES
```

See Appendix B for the syntax and description of these commands. You can write the output from MONITOR to a file by using the SQL*DBA command SPOOL. In MONITOR LOCKS displays, locks appear with the following names:

TM	indicates a <i>data lock</i> on a <i>table</i> . The lock shows the table's ID and may appear in any data lock mode. This lock can be acquired either implicitly or explicitly.
TX	indicates a <i>transaction lock</i> and is acquired for every transaction which has acquired a <i>row lock</i> . It shows the ID of the transaction holding the row lock. Other transactions requesting locks on locked rows must wait for the TX lock to disappear before proceeding.
TD	indicates a <i>dictionary lock</i> on a table. The lock shows the table's ID and may appear in either S mode (a parse lock) or X mode (a DDL lock).
TT	indicates a <i>dictionary lock</i> on a temporary segment.
RT	indicates a redo thread lock.

Note that all locks are not shown in the display. In particular, individual row locks do not appear. However, the appearance of a transaction lock (TX) indicates that a transaction is holding row locks.

Deadlock Detection

A **deadlock** is a situation which can occur in multi-user systems and which causes some number of users to be unable to continue work.

A deadlock can occur when two or more users are trying to access the same database objects. It typically happens when each of two or more users is waiting to access a resource (for example, to lock a table) that another user has already locked. This creates a deadlock situation because each user is waiting for resources held by the other user.

Figure 12-4 illustrates two users in a deadlock.

FIGURE 12-4
Example of a Deadlock
between Two Processes

TRANSACTION A	TIME	TRANSACTION B
Statement 1: UPDATE EMP SET SAL = 1200 WHERE ENAME = 'LEWIS'	X	Statement 2: UPDATE EMP SET SAL = 1000 WHERE ENAME = 'MILLER'
Statement 3: UPDATE EMP SET SAL = 1500 WHERE ENAME = 'MILLER'	Y	Statement 4: UPDATE EMP SET SAL = 1750 WHERE ENAME = 'LEWIS'
		<i>detected interleaved sequence of events</i>

(Handwritten notes: S, R, X, Y, Tx1, Tx2, Deadlock, Rollback preferable, Commit function never releases locks.)

No problem exists at Time X, as each transaction has a row lock on the first row it attempted to update. However, then each transaction proceeds (without committing or ending the transaction in any way) to try to update the row currently held by the other transaction. This causes a deadlock at Time Y, because neither transaction can obtain the resource it needs to proceed or terminate. It is a deadlock because not even waiting will release the lock.

If Statement 3 is rolled back then Transaction B still cannot proceed, as it requires the row locked by Statement 1. Thus Statement 4 is the statement that is rolled back, so that Transaction A can proceed.

When the ORACLE RDBMS detects a deadlock, it signals an error to one of the participating transactions and rolls back the current statement of that transaction. Typically, the statement rolled back is the one belonging to the transaction detecting the deadlock. This resolves the deadlock, although other users may still wait until their response is available. Usually the signalled user should explicitly rollback his transaction but he may simply retry his statement after waiting a few minutes.

Deadlocks often occur when a user escalates his locks. For example, two transactions are both inserting to a table, and thus have row exclusive mode locks. If each transaction then tries to acquire an exclusive lock, say, to read the new data, a deadlock will occur. Because the ORACLE RDBMS itself does no lock escalation, deadlocks occur infrequently in ORACLE.

Avoiding Deadlocks

Deadlocks can always be avoided if users accessing the same tables lock those tables in the same order as each other, either through implicit or explicit locks. You should establish a pre-defined order of access for all tables in your application and then have all applications follow the specified locking order. If this order is followed in all applications, deadlocks will never occur.

When you know you will require a sequence of locks for one transaction, you should consider acquiring the most exclusive (least compatible) lock first.

The INIT.ORA Parameters SERIALIZABLE and ROW_LOCKING

Note: Remaining sections in this chapter are of interest primarily to sites which ran or are running earlier versions of the ORACLE RDBMS or must run in ANSI compatible mode.

When an instance is started, two INIT.ORA parameters determine how it will handle locking: SERIALIZABLE is set to FALSE and ROW_LOCKING is set to ALWAYS for ORACLE with the transaction processing option.

In almost every case, these parameters should not be altered. They are provided for sites who want to use applications written to run with earlier versions of the ORACLE RDBMS; only these sites should consider altering these parameters, as there is a significant performance degradation caused by using other than the default.

The settings for these parameters should be changed only when an instance is shut down. If multiple instances are accessing a shared disk system, instances should usually use the same setting for these parameters.

Setting SERIALIZABLE to TRUE

When SERIALIZABLE is set to TRUE, all locks (and resources) are held at the table level; that is, if you lock one row in a table (with or without the intention of updating it), no one else can modify any rows in that table.

When SERIALIZABLE is set to TRUE, the ORACLE RDBMS schedules all concurrently running transactions to execute such that the result is the same as if each transaction were run one at a time. Users gain the ability to query one table repeatedly throughout one transaction and know they're always seeing the same data (known as "repeatable reads"), *but at considerable concurrency costs*. Although other users can query data you are reading, they must wait until your transaction ends to modify that data.

Setting SERIALIZABLE to TRUE is one way of obtaining "repeatable reads." Three other ways are:

- use SERIALIZABLE=FALSE and lock the table in SHARE mode
- use the SQL statement SET TRANSACTION READ ONLY
- use SELECT FOR UPDATE for all queries, to lock rows in exclusive mode as they are read.

Setting ROW_LOCKING to INTENT

If ROW_LOCKING is set to ALWAYS (the default value for ORACLE with the transaction processing option) and you change data in a table (via INSERT, UPDATE, or DELETE), then other users can update other rows in the same table while your transaction is active.

Since the row level lock manager is available only on ORACLE with the transaction processing option, ROW_LOCKING is always set to INTENT in ORACLE V6 systems without the transaction processing option. When ROW_LOCKING is set to INTENT, then while you are changing data in a table (via INSERT, UPDATE, or DELETE) other users cannot change any data in the table. While you have your lock they can query the data or lock rows with SELECT FOR UPDATE, but they must wait until your transaction is complete to change any data. Thus, while multiple users can have SELECT FOR UPDATE locks on rows of the same table (even in the same block), each user must wait to acquire an exclusive table lock to perform an INSERT, UPDATE, or DELETE.

Summary of Non-Default Locking Options

Because two INIT.ORA parameters are available to change the way locking occurs, there are three global settings other than the default. Figure 12-5 summarizes the non-default settings and why you might choose to run your database system in a non-default way.

FIGURE 12-5
When to Use SERIALIZABLE
and ROW_LOCKING

CASE	I	II	III
SERIALIZABLE =	FALSE	TRUE	TRUE
ROW_LOCKING =	INTENT	ALWAYS	INTENT
Case I:	Equivalent to earlier ORACLE RDBMS versions (no concurrent inserts, updates, or deletes in a table)		
Case II:	ANSI compatible		
Case III:	ANSI compatible, with table level locking (no concurrent inserts in a table)		
	SERIALIZABLE is set to FALSE and ROW_LOCKING is set to ALWAYS in ORACLE with the transaction processing option, which includes row level locking.		
	Figure 12-6 illustrates the difference in locking behavior resulting from the three possible settings of the INIT.ORA parameters as shown in Figure 12-5.		

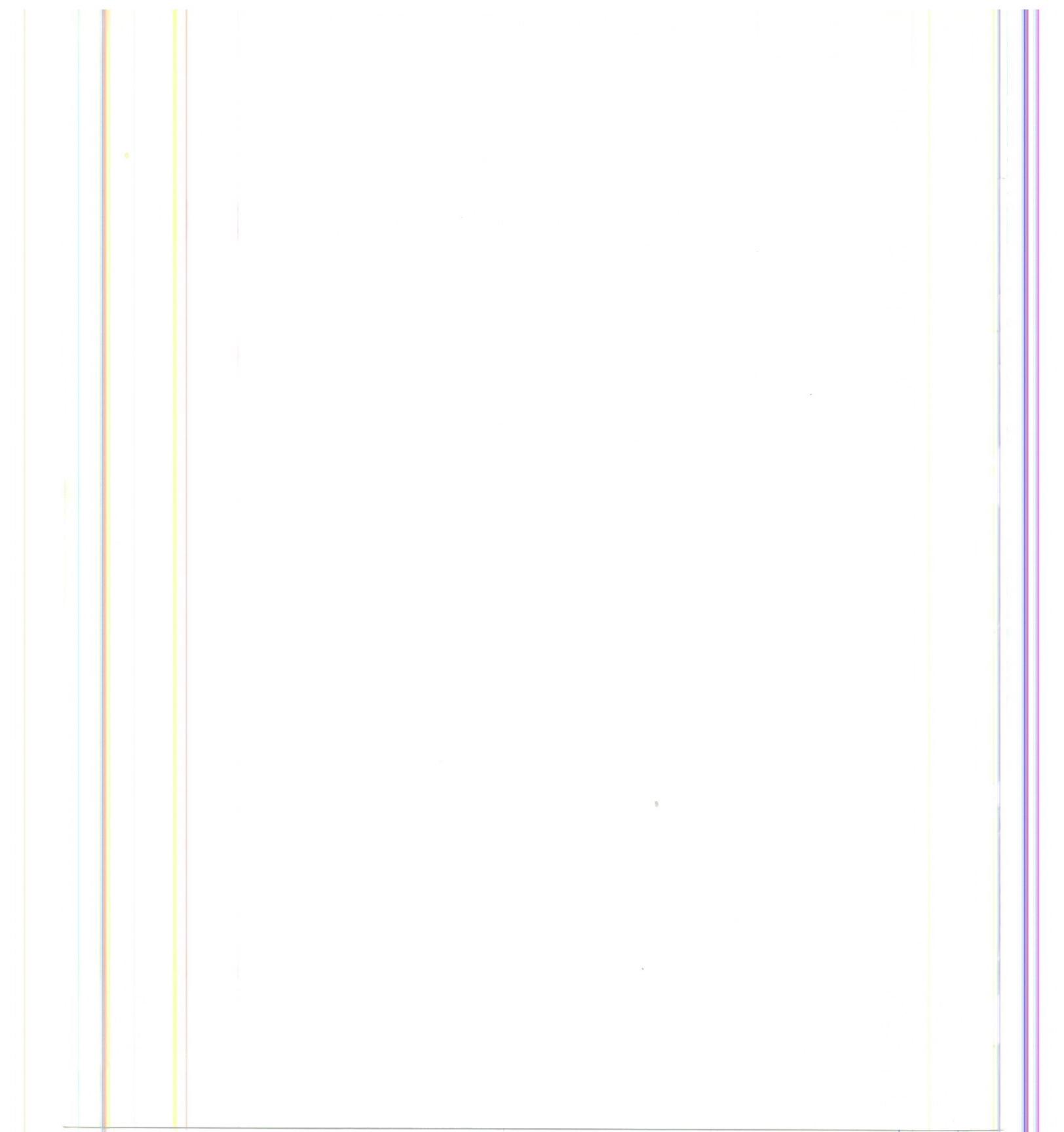
FIGURE 12-6
Non Default Locking Behavior

Statement	Lock Issued (at Row or Table Level)					
	Case I		Case II		Case III	
	row	table	row	table	row	table
SELECT	--	--	--	S	--	S
SELECT FOR UPDATE	X	RS	X	S	X	S
LOCK ... IN xxx MODE:						
EXCLUSIVE	X	X	X	X	X	X
SHARE ROW EXCLUSIVE	SRX	SRX	SRX	SRX	SRX	SRX
ROW SHARE	RS	RS	RS	RS	RS	RS
ROW EXCLUSIVE	RX	RX	RX	RX	RX	RX
SHARE	S	S	S	S	S	S
INSERT	X	SRX	X	RX	X	SRX
UPDATE	X	SRX	X	SRX	X	SRX
DELETE	X	SRX	X	SRX	X	SRX
DDL/DCL	--	X	--	X	--	X

PART

III

PERFORMING DBA FUNCTIONS



13

INITIAL DATABASE CREATION

*Had I been present at the Creation, I would have given some useful hints
for the better ordering of the universe*

Alfonso the Learned

*We are always doing something for Posterity, but I would fain see
Posterity do something for us.*

Joseph Addison

This chapter describes how to create a new ORACLE database. A database must be "created" before users can access it. Database creation means to prepare a database for use by ORACLE and includes:

- installing the ORACLE RDBMS software, if it has not already been installed
- creating new files (or clearing existing files of all prior data)
- writing initial database information into the files
- enrolling initial database users.

Note: Because the information in this chapter applies to databases on all operating systems, it must be relatively general. The *Installation and User's Guides* are the primary source of information about installation; when you actually install and create your database, you should do so using your *Installation and User's Guide*.

In addition, you should also become very familiar with information in Chapter 14 "Database Startup and Shutdown," so you understand the principles and options of starting and shutting an instance, opening a database, and the use of the startup parameter file, typically called INIT.ORA.

Also note that database creation is not necessarily required when moving from one version of the ORACLE RDBMS to another version.

What is Database Creation?

Before a database can be used for data applications, it must be *created*. Creation prepares several operating system files so they can operate as an ORACLE database. A database need be created only once, regardless of how many database files it has or how many instances access it. If the CREATE DATABASE statement is used on an existing database, the data in the database at the time of the second creation is lost.

The act of creation creates new database files, or erases any data which may have existed in the previous database file(s), and writes initial database data which ORACLE requires to access and use the database during normal operation. Database creation also creates and initializes the control file(s) and redo log files for the database.

Actual database creation is performed by a SQL statement CREATE DATABASE; however, before this statement can be executed, the DBA must perform some preliminary steps, as described in following sections.

The Steps for Creating a Database

The basic steps of database creation follow. These steps vary somewhat by operating system and are performed in operating system specific ways. The installation procedure for your operating system may perform many of these steps for you.

Note that creating a database entails the creation and initialization of several files. While file creation is done automatically in some operating systems, other operating systems require that the DBA manually create or allocate files as a first step of database creation.

See your *Installation and User's Guide* to determine the recommended procedure for your operating system. If you must create the files

manually, do so before using CREATE DATABASE with the REUSE option for the control files, data files, and log files.

1. Log on to the operating system account used to install ORACLE.

This account is assumed to be the DBA's account and to have sufficient operating system privileges and access rights to use all SQL*DBA commands, including the privileged commands (see Appendix B). For the required privileges for your operating system, see your *Installation and User's Guide*.

2. Check the INIT.ORA file. Double check its name and the values used for the parameters.

You may choose to use the default INIT.ORA file distributed on the ORACLE RDBMS release media, or you may copy it and edit the copy to better meet your site's requirements. If you do not use the default file, make sure that you identify the file to be used in the STARTUP command using the optional argument PFILE.

Then check that the parameter values are appropriate. Some of the parameters in the INIT.ORA file are critical at database creation; some cannot be changed without re-creating the database. You should check that:

- you have entered a database name of 8 characters or less for the parameter DB_NAME. Although this parameter is optional, it is most convenient to choose the database name at this point. The database name is also used in SQL statements (CREATE and ALTER DATABASE) and the SQL*DBA command STARTUP.
- the values for MAXDATAFILES, MAXLOGFILES, and MAXINSTANCES adequately anticipate the future needs of your database.
- the parameter DB_BLOCK_SIZE is correct. The default database block size is likely to be 2048 or 4096 bytes, depending upon your operating system. You cannot change this size except by re-creating a database.
- CONTROL_FILES provides the names of the *control files*. It may contain either names for new control files, or names of existing control files that may be overwritten. In the second case, use the CONTROLFILE REUSE option for CREATE DATABASE.

Oracle Corporation recommends the use of at least two control files, each located on a different device.

- INIT_SQL_FILES names SQL files that should be run to set up the data dictionary and any initial database objects. The default value is operating system specific and should usually not be altered except possibly to add site-specific files to enhance the data dictionary.

See Appendix D for details on the individual INIT.ORA parameters.

3. Make sure that the default instance is correct.

If you are installing your first database, this step is less critical. However, if you already have any ORACLE databases running, it is important that you do not use CREATE DATABASE on an existing database, or you will lose data. Refer to your *Installation and User's Guide* to see how ORACLE instances are identified and distinguished from each other, so that you are assured that your instance and database are set properly.

4. Enter SQL*DBA by typing the command SQLDBA.

On most operating systems, you simply invoke SQL*DBA by typing SQLDBA.

5. Start up the instance.

Use the SQL*DBA command to start up the ORACLE instance. See Appendix B for complete syntax of the STARTUP command.

```
SQLDBA> STARTUP NOMOUNT
```

If you are using a copy of the distributed INIT.ORA file, use the PFILE=*filename* option:

```
SQLDBA> STARTUP NOMOUNT PFILE=filename
```

Note that neither MOUNT nor OPEN are valid at this point.

6. "Connect" using the keyword INTERNAL:

```
SQLDBA> CONNECT INTERNAL
```

The keyword INTERNAL is used to connect to a database and bypass the RDBMS username checking mechanism. In this case the database has not yet been created (and thus, no data dictionary can perform username validation), so INTERNAL is the only way to connect. SQL*DBA checks that your operating system account or ID has sufficient privileges to connect using INTERNAL. See Chapter 2 for more information on using INTERNAL.

7. Create the new database.

Enter the SQL statement CREATE DATABASE. If you must create the files manually, do so before using CREATE DATABASE with the REUSE option for the control files, data files, and log files. The complete syntax of this statement follows and is described in Appendix G.

```
CREATE DATABASE database_name
[ CONTROLFILE REUSE ]
[ LOGFILE filespec [ , filespec ] ... ]
[ MAXLOGFILES integer ]
[ DATAFILE filespec [ , filespec ] ... ]
[ MAXDATAFILES integer ]
[ MAXINSTANCES integer ]
[ ARCHIVELOG | NOARCHIVELOG ]
[ SHARED | EXCLUSIVE ]
```

All arguments are optional and defaults are used for any argument for which you do not specify values. See your installation guide for your options for *filespec*. You may find it more convenient to include the CREATE DATABASE statement in a file (for example CRDB.SQL) and run the file from SQL*DBA, as in:

```
SQLDBA> @CRDB
```

This step will take longer than each of the others, as many operations are being performed. After completion of these steps you can exit SQL*DBA. The database is created, mounted, and opened, and thus ready for normal access.

Although you can use the ARCHIVELOG argument to CREATE DATABASE in order to create a database in ARCHIVELOG mode, it requires extra overhead and redo log file space to log all the standard steps of database preparation (such as creating and loading the data dictionary). If you are going to use ARCHIVELOG mode (see Chapter 15 for a description of the two modes), it is most efficient to create the database in NOARCHIVELOG mode and then use ALTER DATABASE to change the mode to ARCHIVELOG.

If you do choose to create the database in ARCHIVELOG mode, you should make sure to change the INIT.ORA parameter LOG_ARCHIVE_START to "true" before entering the CREATE command.

What Does CREATE DATABASE Do?

After CREATE DATABASE has the required information, it completes the following steps of database creation, though not necessarily in this order.

Database creation creates the following operating system files.

control files	As many control files are created as are named in INIT.ORA, each one identical to the others. These files should neither be erased nor edited. The use of multiple files guards against damage to a sole control file.
database file(s)	As many database files are created as are specified in CREATE DATABASE; they are all part of the SYSTEM tablespace. If their size is not specified, the default size for the operating system is used.
redo log files	As many are created as are specified IN CREATE DATABASE (there must be at least two). If their size is not specified, the default size for the operating system is used. These files become the online redo log.

Then, database creation prepares the database by:

- creating the SYSTEM rollback segment in the tablespace SYSTEM
- creating and loading data dictionary tables owned by SYS in tablespace SYSTEM by executing the SQL commands in the files named by the INIT.ORA parameter INIT_SQL_FILES
- mounting and opening the database as if the SQL statements ALTER DATABASE ... MOUNT and ALTER DATABASE ... OPEN had been executed.

Depending upon the operating system, these steps may be automatically performed or optional steps of installation.

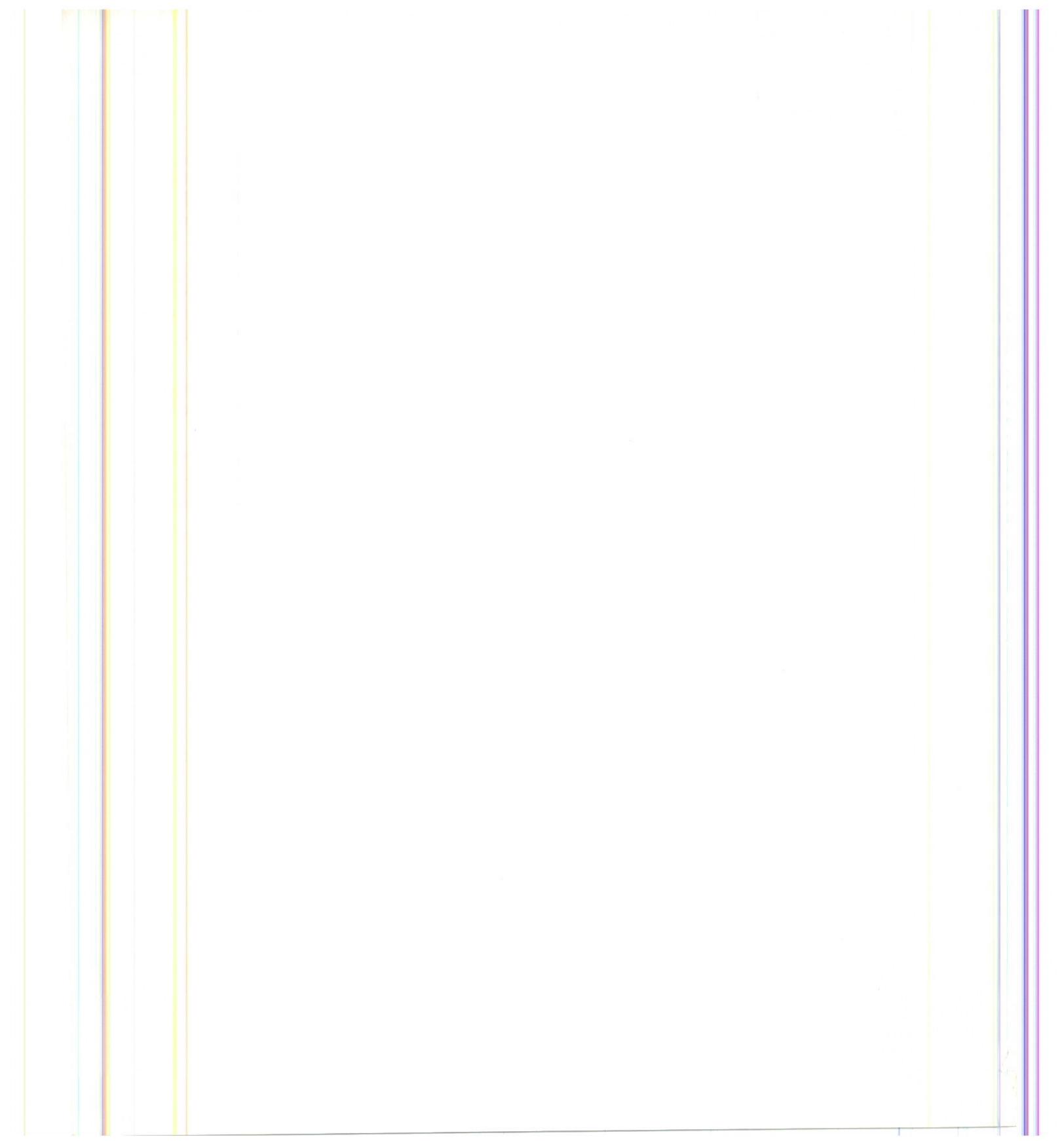
After Creating a Database

After database creation completes, the instance is left running and the database is open and available for normal database use. After database creation you should use the SQL*DBA commands STARTUP and SHUTDOWN to subsequently start and stop the database system.

If you plan to install other ORACLE products to work with this database, refer to the installation instructions for those products; some products require you to create additional data dictionary tables. Refer to the *Installation and User's Guide* and additional documentation for the additional products. Usually command files are provided for this purpose to create and load these tables.

The distribution media may include various SQL files that you may run to experiment with the system, to learn SQL, or to create additional tables, views, or synonyms.

A newly created database has only two users, SYS and SYSTEM. The passwords for these two DBA usernames should be changed soon after the database is created (see Chapter 2). You can enroll new database usernames, with or without DBA privilege, at any time.



14

DATABASE STARTUP AND SHUTDOWN

This chapter describes using the SQL*DBA program to start and stop an ORACLE database system. This chapter presumes your database has already been created using the SQL statement CREATE DATABASE (if not, turn back to Chapter 13). Information in this chapter is intended for DBAs only. Topics include:

- starting up and shutting down instances
- mounting and opening databases
- closing and dismounting databases
- options available for starting systems
- the INIT.ORA file, whose parameters are used to control and tune the performance of a database system.

You may also find the following sections or books useful:

- Chapter 13 Initial Database Creation
- Appendix B The SQL*DBA Command Reference
- Appendix D INIT.ORA Parameter Reference
- The *Installation and User's Guides*

Startup and Shutdown Concepts

An ORACLE database system need not be running whenever the operating system is up, nor does it automatically start unless a site takes specific steps so that it will. Typically a database is started and stopped by a DBA. You can control when ORACLE is running and available, and you have several options when you start or stop a system.

The ORACLE program named SQL*DBA is used to start and stop an ORACLE database system and to perform maintenance and monitoring functions, such as initial database creation, data backup, or media recovery. Instructions for using SQL*DBA and a SQL*DBA command reference appear in Appendix B.

Three steps are required to start an ORACLE database system:

1. Startup an instance.
2. Mount a database.
3. Open the database.

Similarly, three steps are necessary to shut down a database:

1. Close the database.
2. Dismount the database.
3. Shutdown the instance.

Note that two potentially independent units must be combined to form a database system: an *instance* and a *database*. See Chapter 1 for a review of these terms.

Each time SQL*DBA starts an ORACLE instance, it reads a parameter file (typically named INIT.ORA). Then, based on the parameter values, ORACLE creates a system global area (SGA) that will be associated with that instance as long as it is running. Then, the running instance can mount and open a database.

An option allows you to start an instance in DBA mode, so that only users with DBA privilege can access the database.

Mounted vs. Open Databases

Databases exist independently of instances, but must be brought online by an instance in order to be accessed. Two steps are necessary to bring a database online: *mounting* and *opening*. A database must be mounted to be opened, but it can be mounted without being opened. As a result, a database can be in one of three states.

Dismounted	The database is currently not associated with any instance. It is not accessible by users or a DBA until it has been mounted by an instance.
Mounted, but not Open	<p>The database is currently associated with an instance, but it is only accessible by the DBA for certain maintenance purposes. It is not available for normal database operations by users. A database can be mounted in <i>shared</i> or <i>exclusive</i> mode; shared mode is required if the database will be shared by multiple instances, as in shared disk systems.</p> <p>A database must be mounted but not open for a DBA to:</p> <ul style="list-style-type: none"> • add, drop, or rename a redo log file • enable or disable log archiving for purposes of media recovery (switch between ARCHIVELOG and NOARCHIVELOG mode) • perform full database media recovery (rebuild an entire database from backup) • rename database files.
Open	<p>When a database is open it has also been mounted. It is currently available for normal database operations.</p> <p>Only the DBA need be concerned with whether a database is mounted or open; if a user is accessing a database it is always open (and thus mounted).</p> <p>The DBA controls which databases are in which states using the SQL*DBA commands STARTUP and SHUTDOWN, and the SQL statement ALTER DATABASE (with the options OPEN, CLOSE, MOUNT, DISMOUNT).</p>

The SQL*DBA Program

SQL*DBA is an interactive tool used by DBAs (usually via the ORACLE account) to perform the following DBA functions:

- start an instance
- shut down an instance
- open a database
- start an instance allowing DBA access only
- execute any SQL statement
- monitor ongoing use of the database (see Chapter 12 and Appendix B)
- backup database data (see Chapter 15)
- recover database data (see Chapter 15).

All of these operations can be performed on remote as well as local databases. This chapter describes using SQL*DBA for instance startup and shutdown. Appendix B describes invoking SQL*DBA, and contains individual descriptions of the SQL*DBA commands.

Access to SQL*DBA

Because some functions invoked by the SQL*DBA program can be dangerous to the database if misused, few users should have access to use SQL*DBA. The DBA is usually the main user of the SQL*DBA program.

Some SQL*DBA commands are *system privileged* and require special operating system access. These commands are:

```
STARTUP  
SHUTDOWN  
CONNECT INTERNAL
```

Whenever a user invokes one of these commands, SQL*DBA checks whether that operating system account has the proper access to continue. Note that SQL*DBA does not check the ORACLE username because none has been entered; rather, it checks the operating system account currently running SQL*DBA.

Normal database users may have access to SQL*DBA as long as care is taken to restrict access to the system privileged commands. Controlling access to SQL*DBA is discussed in the *Installation and User's Guides*.

Starting an ORACLE Instance and Database

The SQL*DBA command STARTUP combines all startup steps in one convenient command:

```
SQLDBA> STARTUP OPEN dbname
```

However, occasionally it is desirable to execute these steps individually, as follows:

1. Start an instance with SQL*DBA command STARTUP.
2. Mount a database with SQL statement ALTER DATABASE.
3. Open the database with SQL statement ALTER DATABASE.

Descriptions of various startup methods follow.

Startup with Open Database

As just mentioned, to start the current instance and mount and open a database in one command, enter the SQL*DBA command

```
SQLDBA> STARTUP OPEN dbname
```

where *dbname* is the name of the database to be opened. Alternatively, if the database name is specified by the INIT.ORA parameter DB_NAME, you can simply enter:

```
SQLDBA> STARTUP OPEN
```

The specified database must already have been created. (The instance identifier is assumed to be specified; the method varies and is described in the *Installation and User's Guides*.)

The STARTUP command used with the OPEN option:

1. Starts an ORACLE instance:

- creates and initializes the SGA, using the current INIT.ORA file parameter values
- starts background processes.

2. Mounts the named database:

- finds and opens control files.

3. Opens the mounted database:

- finds and opens redo log files and online database files
- if required, performs rollforward (redo) recovery
- enables logging to the online redo log files
- if required, performs rollback (transaction) recovery.

When recovery completes, the database is open and available for normal use by all database users.

An example of this type of startup follows:

```
SQLDBA> STARTUP
ORACLE instance started.
Database mounted.
Database opened.
Total System Global Area 372520 bytes
Fixed Size          12312 bytes
Variable Size       286480 bytes
Database Buffers   65536 bytes
Redo Buffers        8192 bytes
```

Note that the size of the SGA is displayed upon startup.

Mounting and Opening in Separate Steps

You may wish to start an instance without mounting or opening a database in order to do these steps individually. This may be desirable when performing maintenance tasks. The following sequence of commands describes the individual steps; usually the DBA will also execute some intervening commands, depending on his intended tasks.

1. First, start the instance:

```
SQLDBA> STARTUP NOMOUNT
```

2. To mount a database, you must connect as INTERNAL:

```
SQLDBA> CONNECT INTERNAL
```

3. Mount the database. You must specify a *dbname* at mount time.

```
SQLDBA> ALTER DATABASE dbname MOUNT
```

4. Open the database. You need not specify a *dbname* because the database currently mounted is assumed.

```
SQLDBA> ALTER DATABASE OPEN
```

Startup for DBAs Only

Access to a database can be limited to DBAs only. For example, if you need to delete a redo log file or change the parameters in the INIT.ORA file, you can stop the database system at a low-use time to do so.

To start a database system which can only be accessed by DBAs, enter:

```
SQLDBA> STARTUP DBA OPEN dbname
```

When you want to allow users access again, just shut down and restart the system without the DBA argument.

Startup with FORCE Option

Use the FORCE option if you wish to shutdown a currently running instance as with SHUTDOWN ABORT and then restart the instance. This option is used infrequently but can be used to shut and restart an instance quickly and using one command.

Startup with Tablespaces Offline

When a database is opened, all tablespaces are online or offline, as they were when the database was last closed. Tablespaces may be taken offline after a database is opened using the SQL statement:

```
ALTER TABLESPACE tablespace OFFLINE [ NORMAL | IMMEDIATE ]
```

See Chapter 16 and Appendix B for a more complete description of this command. For more information on offline tablespaces, see Chapter 4.

To bring the tablespace back online, enter:

```
ALTER TABLESPACE tablespace ONLINE
```

Automatic Startup with Operating System Start

Many sites use procedures to enable automatic startup of one or more ORACLE instances and databases immediately following a system start. The procedures for doing this are specific to each operating system; refer to your *Installation and User's Guide* for more information.

Starting Remote Instances

Procedures for starting and stopping remote instances vary widely depending on communication protocol and operating system. The following are very general guidelines (currently VMS based) for starting a remote instance.

1. At the remote node, create a file to set up the proper configuration for the database file (to point to the proper control file(s), database files, INIT.ORA file, and redo log files).
2. At the local node, define a logical for the driver string to login remotely. The remote account must have adequate privileges on the remote node to startup the database. SQL*Net will login at the remote node through this account and execute the file created in Step 1 to point to the proper database.
3. At the local node, begin a SQL*DBA session.
4. Use the SQL*DBA ommand SET INSTANCE with the driver specified in Step 2:

```
SQLDBA> SET INSTANCE driverstring
```
5. To startup the database, enter the STARTUP command:

```
SQLDBA> STARTUP ...
```

If you don't specify the remote INIT.ORA file, the startup will use the local INIT.ORA file.

6. To shutdown the remote instance, simply use the SHUTDOWN command:

```
SQLDBA> SHUTDOWN
```

The INIT.ORA Parameter File

Every time SQL*DBA starts an ORACLE instance, it uses a set of parameters which specify characteristics of that instance's operation. The parameters are found in a file typically named INIT.ORA (or some variation, such as INIT`dbname`.ORA).

The INIT.ORA file contains several types of parameters, the most important of which directly affect the performance of an ORACLE database system. For example, it contains parameters which must have values in order to start an instance (such as the names of the control files), and parameters that set the number of various entries in the SGA. By adjusting the latter type of parameter, you can increase or decrease the size of the SGA (space in memory) and affect ORACLE's performance.

Using Alternate Names for INIT.ORA

The parameter file need not be named INIT.ORA. If the filename is different than your system's default name, it must be specified using the STARTUP argument PFILE, as in:

```
SQLDBA> STARTUP PFILE=OURPARM.ORA
```

Where SQL*DBA expects to find the file is operating system dependent (refer to the *Installation and User's Guide* for your operating system). In this manual we refer to the INIT.ORA file even though you might use a file with another name.

Sample INIT.ORA File

A sample INIT.ORA file is included with the distribution media for ORACLE. This file's parameters are adequate for most initial installations of an ORACLE database. After your system is operating and you have some experience with ORACLE, you will probably want to experiment with changing some parameter values.

A sample INIT.ORA file is shown in Figure 14-1.

FIGURE 14-1
A Sample INIT.ORA
Parameter File
for DEC VAX/VMS list

```
db_name=PROddb
log_checkpoint_interval = 10000
row_cache_enqueuees = 200
sessions = 50
calls = 50
processes = 50
ddl_locks = 200
dml_locks = 200
control_files = disk$dev2:[oracle]control1.ora
```

Groups of INIT.ORA Parameters

The INIT.ORA parameters are individually described in detail in Appendix D. Most INIT.ORA parameters fall into one of the following groups:

- parameters that name things (such as files)
- parameters that set limits (such as maximums)
- parameters affecting capacity, called *variable parameters*.

Because they impact performance and have the most flexibility, the variable parameters are probably of most concern to DBAs.

Why Change Parameters?

You may adjust variable parameters to improve the performance of a database system. Exactly which parameters will most benefit your system is a function of numerous database characteristics and variables. Many methods of tuning data schemas, application design, operating system parameters, and so on, are documented elsewhere in this manual or in other Oracle documentation. As with tuning operating systems, before changing parameters you should measure current performance, understand the significance of the parameter being changed, expect a certain result, and measure the result to see if the gain was achieved.

Other suggestions for tuning your ORACLE database are found in Chapter 20. Also refer to your *Installation and User's Guide* for operating system specific tuning hints.

Shutting Down an ORACLE Instance and Database

Shutting down a database system entails the following steps:

1. Close the database with SQL statement ALTER DATABASE.
2. Dismount the database with SQL statement ALTER DATABASE.
3. Shut down the instance with SQL*DBA command SHUTDOWN.

These may be combined in one step using options to the SHUTDOWN command.

Normal Shutdown

The most common method of shutting down an instance is to simply enter:

```
SQLDBA> SHUTDOWN [NORMAL]
```

SHUTDOWN entered with no options is the same as SHUTDOWN NORMAL. This type of shutdown:

- waits for currently connected users to disconnect
- prohibits new connects
- closes and dismounts the database
- shuts down the instance.

A startup after a normal shutdown requires no instance recovery (rollforward or rollback).

If SHUTDOWN NORMAL does not work due to database problems, you can enter a stronger SHUTDOWN command.

Shutdown IMMEDIATE

SHUTDOWN IMMEDIATE should be used when you do not want to wait for current users to log off.

```
SQLDBA> SHUTDOWN IMMEDIATE
```

This type of shutdown is identical to a normal shutdown, except that:

- current calls are terminated as if by a system interrupt
- it does not wait for currently connected users to disconnect.

The background process PMON terminates all current user sessions by performing a rollback, and the database is closed and dismounted.

Application programs receive an error message from ORACLE which can be handled in any way appropriate for the application.

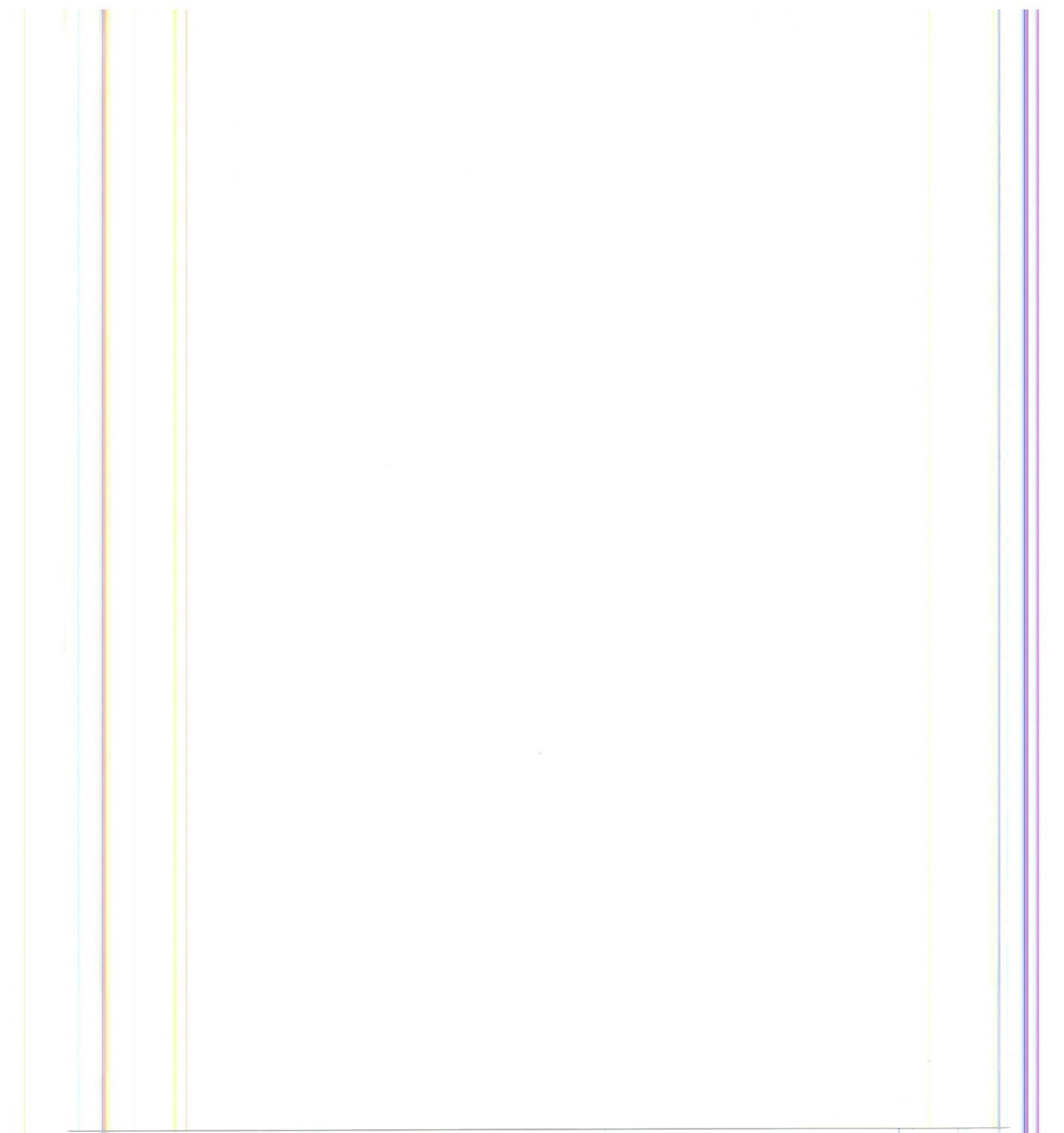
A startup after a SHUTDOWN IMMEDIATE does not require instance recovery (rollforward and rollback).

Shutdown ABORT

SHUTDOWN ABORT is the most drastic and fastest shutdown available. It does not wait for current calls or connections to complete, but shuts down the instance as quickly as possible.

Shutdown ABORT should be used when either one of the other SHUTDOWN options fails, in which case the instance may not be running properly. It is not necessary to try SHUTDOWN IMMEDIATE first, if SHUTDOWN [NORMAL] did not work.

A startup after a SHUTDOWN ABORT will perform instance recovery (rollforward and rollback) if it is required.



15

DATABASE BACKUP AND RECOVERY

Character is much easier kept than recovered.

Thomas Paine, *The American Crisis* No. XIII (1783)

This chapter covers the following topics regarding database backup and recovery:

- types of failure requiring recovery
- features of the ORACLE RDBMS designed for data backup and recovery
- preparing for recovery by taking various types of backups
- using the database in ARCHIVELOG and NOARCHIVELOG mode
- recovering from various failures
- using SQL*DBA commands ARCHIVE and RECOVER
- the role of the Export and Import utilities in a backup strategy
- hints on choosing recovery strategies.

You may also find the following references useful:

- Chapter 3, File Structure, describes the redo log.
- Chapter 4, Tablespaces and Segments, describes rollback segments.

- the *Installation and User's Guides* contain detailed operating system information about backup and archiving of redo log files
- the *ORACLE Utilities User's Guide* describes the Export and Import utilities
- Chapter 21 discusses the additional recovery needs of a shared disk system.

Different types of failures require different precautions or recovery steps. This chapter examines the tradeoffs between protecting your database from various types of failures and the associated costs of operation and administration.

ORACLE provides some utilities specifically designed for backup and recovery of ORACLE data. Some of these require standard backup features offered by host operating systems or third party software, such as full disk backup.

Why is Recovery Important?

A major responsibility of the DBA is to prepare for the possibility of system failure. Should system failure occur, the database must usually be recovered as quickly and with as little detrimental impact on users as possible.

Recovering from any failure requires a DBA to:

1. Determine what data structures are intact or need recovery.
2. Follow the appropriate recovery steps.
3. Restart the instance so that it can resume normal operations.
4. Assure that no work is lost or data incorrect in the database.

The goal is to return to "normal" as quickly as possible, and at the same time insulate database users from any problems or the possibility of losing or duplicating work.

The recovery process varies depending on the type of failure and the portion of the database affected by the failure. If the database files are found to be intact, recovery may be as simple as restarting an instance.

You should anticipate each type of failure and have a recovery strategy which will handle each failure. In choosing recovery strategies, you face tradeoffs, primarily between recoverability and administrative requirements. In general, the more protection desired, the higher the cost in DBA responsibilities.

ORACLE Recovery Features

The ORACLE RDBMS offers several features to provide flexibility in recovery strategies:

- recovery from system, software, or hardware failure
- automatic instance recovery at database startup
- online backup and restoration of single tablespaces or files, while the rest of the database is operational
- optional media recovery protection
- minimal incremental administrative tasks for preparing for media failure
- increased control over recovery time in the event of system failure
- Export and Import for archiving and restoring data

Types of Failure Requiring Recovery

Several circumstances can halt the operation of an ORACLE instance or affect the writing of data to the database. Some examples are access failures to disks containing the ORACLE database files and operating system errors accessing ORACLE program files.

Some types of failure are described below. In some cases, recovery is automatic and requires little or no action on the part of the database user or DBA.

User Error

A DBA can do little to prevent user error, but an effective recovery scheme may ease recovering from many user errors. For example, if an incremental export is taken nightly and a user accidentally drops a table that had existed longer than a day, that table will have been saved on one of the incremental export files. In most cases, user error can be reduced by increased training of database and application principles.

Statement Failure

Statement failure occurs when there is a logical failure in the handling of a statement in an ORACLE program. The ORACLE software or operating system will return an error code or message. This type of failure usually requires no action on the part of the DBA.

When statement failure occurs, the effects of the statement are automatically rolled back, before control is returned to the user or user program. Depending on the situation, the application program may be

written to anticipate and respond to errors, or the user can simply re-execute the statement after correcting for the problem. However, sometimes it is appropriate for the DBA to alter an INIT.ORA parameter, if the error occurs repeatedly and can be avoided by increasing database resources.

Process Failure

A process failure is a failure in a user process accessing ORACLE (for example, an abnormal disconnect, termination, or addressing exception). The process cannot continue work, although the instance can.

The ORACLE background process PMON detects the aborted process, and resolves the problem by rolling back the process's transaction and releasing any resources the process was using. This process is automatic and requires no intervention by the DBA.

Instance Failure

Instance failure occurs when a problem arises which prevents an instance from continuing work. The problem may be due to a hardware problem, such as a power outage, or to software failure, such as an operating system crash. An instance failure can also be caused by ORACLE RDBMS software failure; the most common symptom is when one or more of the background processes (DBWR, LGWR, ARCH, SMON, or PMON) terminates abnormally.

Instance failure requires *instance recovery*. This may require action on the part of the DBA but it is usually accomplished automatically using files already on disk.

In a *non-shared disk system* or if all shared-disk instances fail, the DBA need only shut down and restart the instance; instance recovery is performed automatically every time an instance is started.

In a *shared disk system*, when one instance fails and at least one other instance is still running, another instance will perform recovery for the failed instance (see Chapter 21).

Media Failure

A physical, non-recoverable error can arise when trying to write or read a file required to operate the database. An example is a disk head crash causing the loss of all files on a disk drive. Several files may be affected by a media failure, including the log files, the control files, and database files (corresponding to the SYSTEM tablespace or other tablespaces). Data recovery strategies vary depending on which files are affected.

Basic Recovery Steps

Data recovery requires two explicit types of actions:

- | | |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rolling forward | to verify that transactions that committed and changed database data have been written to the database. If necessary, a transaction's operations are re-applied. |
| rolling back | to verify that transactions that rolled back or never committed have been "erased" from the database. If necessary, a transaction's operations are undone. |

As transactions are processed during database operation, many database blocks may be modified. Rather than directly changing the actual database blocks, the RDBMS uses a temporary workspace called the *database buffer pool*. Because the buffers contain committed or uncommitted data, they allow users to rollback work easily and allow the database to support many concurrent read-consistent views of the same tables.

Because the number of database buffers is limited, they are reused, requiring that their contents be written to disk. To reduce I/O, buffers are written to disk only when necessary, using the "least recently used" (LRU) algorithm. Buffers that have not been used for the longest time are written to the database.

Thus, the most recently changed database blocks are stored in the database buffers, whether or not their contents have been committed. And, the least recently changed database blocks are written to the database, whether or not their contents have been committed.

When a transaction terminates normally (commits), the redo log records the transaction's changes synchronously with the commit. That is, a user's transaction is made permanent with a single write to the redo log file. DBWR writes the changed blocks from the database buffers to the database according to the LRU algorithm. Normal transaction termination also verifies that any data rolled back is undone in the database buffers and actual database files.

Note that at any given time the database may contain some blocks modified by uncommitted transactions; this is due to the LRU algorithm used by DBWR. And, at any given time, the database may not contain all blocks corresponding to committed transactions; this is because DBWR writes asynchronous to commits. Two potential situations may result:

- The database may contain data which was not committed because the modifications were written when the buffers were the least recently used. This data must be erased from the database either on warmstart or recovery.
- Since modified blocks are not written to the database at commit time but appear only in the redo log, the redo log may contain committed data that is not in the database and which must be applied to the database.

The ORACLE RDBMS uses two types of structures to resolve these issues:

Rollback segments Rollback segments are used for *rolling back* transactions (data which was not committed). See Chapter 4 for a description of rollback segments.

Redo log The redo log is used for *rolling forward* transactions (data which was committed but not yet written to the database). See Chapter 3 for a description of the operating system files which comprise the Redo Log; this chapter describes how the Redo Log works.

Rolling Forward with the Redo Log

The redo log is a set of operating system files external to the database that record all changes made to the database (committed, rolled back, or uncommitted). During recovery, all changes in the redo log are applied to the database. The "redoing" of all previously entered changes is called *rolling forward*.

Rolling forward may entail applying only entries from the online log files, or it may require the restoration of older offline redo log files stored on secondary storage devices, typically magnetic tapes.

Rolling forward proceeds through redo log files to bring the database forward in time. After rollforward, the database blocks contain all committed changes as well as any uncommitted changes from transactions in progress at the time of the failure.

Rolling Back with the Rollback Segments

The roll forward is only half of recovery; after the roll forward, any changes which were not committed must be undone. After the redo log files have been applied and all changes made to the database, then the rollback segments are used.

Rollback segments are portions of the database that contain information sufficient to undo any uncommitted changes to data. The rollback segments are used to identify and undo transactions that were active at the time of a failure. This process is called *rolling back*.

(Note that because the rollback segments are contained within the database, they too are protected against media failure by the redo log and are recovered during the rollforward process.)

Rollback recovery is automatic and always occurs when an instance is started as a part of instance recovery. ORACLE identifies which rollback segments contain uncommitted data. You need not take any additional action for rollback recovery.

After both the log and rollback segments have been applied, the database is restored to a consistent state.

Instance Recovery

Instance recovery is performed automatically when required upon instance startup (usually after a SHUTDOWN ABORT, and never after a SHUTDOWN IMMEDIATE or NORMAL). It requires no special DBA action other than starting an instance.

Instance recovery is also required when an instance cannot continue work. Common causes of instance failure include:

- CPU failure
- operating system crash or reboot
- power failure
- failure to access a required database file
- failure of one of the ORACLE background processes.

If you notice that an instance has suspended work, stop the ORACLE instance with the SQL*DBA command SHUTDOWN, using the IMMEDIATE or ABORT options if necessary. (A NORMAL shutdown is preferred because it reduces the time required for the next startup.) Then restart the instance in the normal manner, using STARTUP. Restarting the instance begins the instance recovery process.

Instance recovery restores a database to its state just prior to failure by:

- rolling forward using the online redo log files, to restore committed data that had not been recorded in the database, to restore uncommitted transactions whose changes were recorded in the log, and to restore the contents of the rollback segments
- using the rollback segments to roll back transactions which were either explicitly rolled back or not yet committed
- releasing resources held by the transactions.

If you often experience instance failure due to the failure of one of the background processes, you should look for trace (dump) files generated by that process. These files are found in the location specified by the INIT.ORA parameter BACKGROUND_DUMP_DEST and may give some indication of why the process is failing.

Media Recovery

Media failure occurs when a file or device either cannot be read or cannot be written to, because it is damaged or missing. Failure when reading and writing files can frequently be traced to temporary problems with operating system hardware or software; in these cases no media recovery is necessary.

However, when you know that a file related to the database has been damaged or lost, then you must either perform media recovery or restore your database to a recent backup (as in a full database export or image backup).

To perform media recovery you must:

- have backup copies of the damaged database files
- use the redo log in ARCHIVELOG mode,
- have archived (offline) redo log files starting at the point of backup.

Media recovery of the database involves the same steps as instance recovery (roll forward and roll back). The primary difference is that media recovery may need to start at an earlier point in time and thus may require offline (older) redo log files.

If you want to be able to recover from media failure with no loss of work, you must use ARCHIVELOG mode, saving redo log files in offline storage. If you are willing to re-enter work done since a backup and before a media failure, then you can restore a saved copy of the database taken when no instance is using the database, and then reenter your work.

Media failure can occur in four distinct parts of an ORACLE database system:

- a database file that is not in the SYSTEM tablespace
- a database file that is in the SYSTEM tablespace
- an online redo log file
- a control file.

Procedures for remedying these situations are described later in this chapter. In the simplest case, as in the loss of one control file, you may simply need to copy a file. In the first two cases, recovery involves recovering the tablespace containing the file or the entire database. The SQL*DBA utility is used to apply online and offline redo log files to restored copies of database files.

Database Restoration

A final type of recovery is simply restoring the database to its state as of a particular file archival, in which all database files were archived. It is a good procedure to periodically back up a consistent set of database files while the database is offline. A *consistent backup*, taken when the database is offline, includes all database files, all online log files, and all control files. Such a backup could be used, for example, in the case of loss of a redo log file.

How the Redo Log Works

The redo log can be used in two modes: ARCHIVELOG or NOARCHIVELOG. The mode is chosen when CREATE DATABASE is invoked, and can be switched with ALTER DATABASE any time the database is closed.

All changes made to the database are logged, regardless of the mode chosen. Furthermore, information necessary for instance recovery can always be found in the online redo log, regardless of the mode chosen.

Questions you should ask to determine which mode to operate in include:

- Do you need to prepare for the possibility of full media recovery?
- Do you need to perform online backups, with the system running, to provide continuous availability?
- Do you want to assume the extra DBA responsibilities associated with archiving log data (such as providing for storage, record keeping and so on)?

If you want to use the additional recovery options, you must consider the price of the incremental administration, such as keeping track of files and tapes and doing regular backups. The benefit of this additional work is that you will be able to restore your database and lose no work, in the event of any type of hardware or software failure. And, you will be able to take database backups while the database is in use (online backups require the archiving of redo log files).

Enabling archiving costs your database system nothing in performance, since ORACLE writes to a redo log file regardless of the mode chosen. The only additional work is copying of the redo log files to offline storage and performing periodic backups of the database.

Using the Log in ARCHIVELOG Mode

In ARCHIVELOG mode, all changes to the database are saved so that they can be reconstructed in the event of media failure. ARCHIVELOG mode requires the use of both an online redo log and an offline redo log and periodic physical backups of the database.

The backup must be physical (as from a disk or image backup) rather than logical (as from an export) and it must contain all files related to a given tablespace. These backups may be performed while the database is open for normal use. Frequent backups are preferred; the older the backup, the more log files are required to recover.

Then, in the course of normal database operation, the redo log records all transactions following that backup until the next backup is taken. These transactions are written to the online log files, and, as those files fill, are archived to offline log files.

The Online Redo Log

The *online redo log files* exist regardless of whether you are using ARCHIVELOG or NOARCHIVELOG; they are the set of operating system files currently being written to by the ORACLE RDBMS. Only the online log is necessary to accomplish instance recovery.

When an online log file fills, the data it contains must be saved offline (usually written to tape) before that particular online redo log file can be reused. ORACLE prevents you from overwriting an online log file in ARCHIVELOG mode until it has been archived. The archiving of data from the online redo log file to an offline redo log file can be *automatic* or *manual* (described later in this chapter).

The Offline Redo Log

The *offline redo log files* are present only for ARCHIVELOG mode. The offline log is simply the collection of all redo log files that have been archived. The offline log is typically not immediately accessible but is stored offline and may require some action to bring it online. Adequate

storage (usually offline) is required to store all changes which occur in each interval between backups.

Offline redo log files are identified by the *log sequence numbers* assigned to them when they are archived. These numbers begin with 1 and are unique for each file. The DBA is responsible for tracking the association between each offline redo log file and its log sequence number; the sequence numbers are used during recovery.

Using the Log in NOARCHIVELOG Mode

In NOARCHIVELOG mode, the redo log files are written exactly as in ARCHIVELOG mode, but a redo log file can be reused without saving its contents offline. The online log files are used in a "circular" fashion (when the last online redo log file "fills" the ORACLE RDBMS "wraps" to the first file and begins overwriting its contents), so no offline log is necessary.

Only the most recent changes to the database are available at any time. This has three major implications:

1. The redo log can only be used to recover from failures that occur while the necessary redo log files are still online. While NOARCHIVELOG always allows you to recover from instance failure, it usually does not allow recovery from media failure because not enough roll forward data has been saved.
2. If a tablespace becomes unavailable due to some failure, you cannot continue to operate the database until the tablespace has been recovered or dropped, because to do so might overwrite data in the online redo logs that is required for the recovery of that tablespace.
3. You cannot conduct online backups; you may only backup the database when it is shutdown.

Checkpoints

Whenever a transaction is committed, its corresponding redo entries are written from the redo log buffers to the online redo log. Occasionally redo entries are written to the log before their transaction is committed if the redo log buffer pool fills or if those entries are in the same buffer as entries from a committed transaction.

Checkpoints occur when all the modified database buffers are written to the database files, to make sure that all changes have been written to disk, with the effect that earlier log entries are no longer needed for instance recovery. Checkpoints are initiated and accomplished automatically.

By periodically saving the changes to the database, checkpoints serve two purposes:

- decrease the time required by instance recovery if it should be required
- allow an online redo log file to be archived or reused.

If a second checkpoint is initiated while a first one is in progress, the first checkpoint stops so the second can begin. The second checkpoint performs the work for both.

A checkpoint consists of remembering the location of the next entry to be written in the redo log file, writing the modified database buffers to disk, and then writing the remembered location to the control file and database files. Then, if recovery is necessary at some time after the checkpoint is taken, it can start at the remembered location.

Though some overhead is associated with a checkpoint, the database system does not halt activity nor are current transactions affected. Because DBWR continuously writes database buffers to disk, a checkpoint does not necessarily require many database blocks to be written all at once. Rather, the completion of a checkpoint simply insures that all blocks modified since the previous checkpoint are actually written to disk.

Though all checkpoints have the same effect, they may be initiated in two ways, as described in the next two sections. You cannot force a checkpoint to occur, but you can control the frequency at which they occur. The more frequently they occur, the less time will be required by instance recovery.

Checkpoints for Speeding Recovery

Checkpoints occur after a specified number of redo log blocks have been written and serve as an intermediate point for recovery. All changes prior to the checkpoint are "safe," so recovery can resume with only those changes following the checkpoint.

To control when these checkpoints occur, set the number of redo log blocks for the INIT.ORA parameter LOG_CHECKPOINT_INTERVAL.

Smaller values for this parameter reduce the time necessary for recovering from instance failure, but increase the frequency of checkpoints being taken and have a negative impact on performance (since each checkpoint requires extra I/O on the part of DBWR).

Checkpoints When a Redo Log File Fills

A checkpoint is also used to allow online redo log files to be reused. These checkpoints occur whenever a redo log file fills, and while writing the next redo log file. In NOARCHIVELOG mode, a checkpoint must simply complete before the redo log file can be reused. In ARCHIVELOG mode, a checkpoint must occur and the log file must be archived before the file can be reused.

DBA Decisions Regarding Recovery Structures

The DBA has several choices regarding preparing for recovery, including decisions about the following recovery structures:

- redo log files
- rollback segments
- control files.

The following sections discuss considerations regarding these structures; refer also to Chapters 3 and 4 for additional information.

Creating the Online Redo Log

The online redo log always contains at least two operating system files. When you create the redo log (at the same time you create the database) with the command CREATE DATABASE, you can specify:

- whether to use ARCHIVELOG or NOARCHIVELOG mode
- the number of online redo log files
- the size of the files
- where to place the files.

You may also specify and alter the following at any time:

- the size of the LOG_CHECKPOINT_INTERVAL
- for shared disk systems only, the size of the LOG_ALLOCATION for each instance (see Chapter 21)
- file specifications for online redo log files.

These decisions affect how often checkpoints will occur and, if you are using ARCHIVELOG, what requirements or flexibility you will have regarding archiving.

The original redo log files are created with the execution of CREATE DATABASE. The syntax allows for the specification of two or more log files:

```
CREATE DATABASE ... [ LOGFILE filespec [, filespec] ]
```

where *filespec* is:

```
filename [SIZE integer] [REUSE]
```

If you use neither SIZE nor REUSE, REUSE is implied. Size is in bytes, as in:

```
SQLDBA> CREATE DATABASE TESTDB  
LOGFILE 'ONELOG.RDO' SIZE 50000, 'TWOLOG.RDO' SIZE 50000;
```

If you omit the optional argument LOGFILE, two small redo log files are automatically created to allow the database to be opened. The names, sizes, and placement of default redo log files created in this manner vary by operating system; see the *Installation and User's Guide* for details specific to your operating system.

Choosing the Redo Log Mode

In the CREATE DATABASE statement you may specify one of two redo log file options: ARCHIVELOG or NOARCHIVELOG. If neither is specified, NOARCHIVELOG is the default.

The mode can later be changed when the database is mounted but not open, using ALTER DATABASE.

Choosing the Number of Log Files

The online redo log must have at least two log files allocated to it; it may have more. The question is really twofold:

- how much total log space will you require?
- how do you want it divided among files?

Although the minimum number of redo log files is two, the use of more than two is recommended. Using more files can add some flexibility to your spooling strategy. For example, if you want 10,000 blocks of redo log space, and you want to spool to 2,000 block tapes, using 5 files of

2,000 blocks each allows you to spool a file to tape each time any file fills.

The maximum number of redo log files which can ever be used is MAXLOGFILES, as specified in the CREATE DATABASE statement. The maximum number at any given time is the number specified by the INIT.ORA parameter LOG_FILES. This parameter can temporarily override MAXLOGFILES to a lower number; it cannot increase the limit. The default for both is 16 and the absolute maximum is 256.

Setting the Size of the Redo Log Files

Bearing in mind how much total redo log space you require, you then must decide on the desired number of files and the desired size of each. Following are some guidelines for setting the size of online redo log files.

Each redo log file should be a minimum 50 Kbytes; there is no maximum size. Redo log files need not be the same size as each other but usually will be for convenience.

If you have larger online redo log files, then:

- it will take longer to fill each file
- fewer checkpoints due to switching files are required
- you will need to archive online files to offline storage less often
- instance recovery may take slightly longer unless you reduce the checkpoint interval.

Setting Devices for Redo Log Files

Redo log files should be placed on different storage devices from the ones containing the database files for two reasons:

- to reduce the risk of losing all database and redo log files in the event of media failure
- to reduce disk contention (during times of high database activity the log file is frequently accessed).

Only one redo log file is written to at any time, so it is not advantageous to spread redo log files across disks.

Altering the Log

You can change the configuration of the redo log by:

- adding new redo log files
- dropping redo log files
- renaming redo log files
- switching between ARCHIVELOG and NOARCHIVELOG.

Changes to the configuration and use of the redo log must be done while the database is mounted exclusively by one instance, but not open by any instances. See Chapter 16 for more information.

Changing between NOARCHIVELOG and ARCHIVELOG Mode

You can switch the mode of the redo log using the ALTER DATABASE statement. To do so, the database must be mounted exclusively but not open. You must be connected as INTERNAL in order to execute the SQL statement ALTER DATABASE against a closed database.

If you are switching from NOARCHIVELOG to ARCHIVELOG, then you should backup (not export) all database files immediately after the switch. The database will not be recoverable until this backup has occurred. You need not backup redo log files or control files.

If you are switching from ARCHIVELOG to NOARCHIVELOG, then media recovery is limited to a roll forward to the time of the switch. Only instance recovery is possible thereafter.

Choosing a Checkpoint Interval

The *checkpoint interval* is the number of redo log blocks used as a threshold for triggering a checkpoint. The value is set by an INIT.ORA parameter LOG_CHECKPOINT_INTERVAL. The default is operating system dependent; there is no maximum or minimum. If the interval is larger than the size of a redo log file, then checkpoints will occur only on switches between online log files.

If you reduce the interval, then:

- checkpoints occur more frequently
- the time required for instance recovery at startup will decrease (because fewer blocks of the redo log must be applied to the database to make it current).

Creating Rollback Segments

During CREATE DATABASE, an initial rollback segment is created using default settings. One segment is adequate for small initial systems (only one tablespace) or low-use systems. However, most DBAs will want to add rollback segments to accommodate more users and more transaction activity, or to allow multiple tablespaces or shared disk operation.

To add new rollback segments, use the SQL statement:

```
CREATE ROLLBACK SEGMENT
```

See Chapter 16 for instructions on adding rollback segments. You must shutdown and restart a system in order for a rollback segment to be used.

DBAs typically have the following decisions regarding rollback segments:

- How many rollback segments to use?
- How should space be allocated for each segment?
- Should each rollback segment be public or private?
- In which tablespaces should the segments reside?

In making these decisions, you should consider:

- How much database space should be or can be dedicated to rollback segments?
- What is the average and maximum number of concurrent transactions?
- What are your performance requirements?
- Can you distribute rollback segment I/O across devices?
- Are you running a single instance or in shared disk mode (multiple instances)?

Choosing the Number of Rollback Segments

The total number of rollback segments *required* by a database system depends on:

- the maximum expected number of concurrent transactions
- the number of transactions allowed per rollback segment
- for shared disk systems, the number of instances accessing the database

Increasing the number of rollback segments reduces contention for rollback segment blocks. If you run MONITOR ROLLBACK SEGMENTS and see a high number in the column *Header Waits/Sec*, you should add more rollback segments to reduce contention.

Rollback segments maintain transaction information for many purposes, including read consistency. A rollback segment can accommodate a limited number of transactions, called the number of *transaction slots*. The number is a function of information stored in the rollback segment about active transactions. When a transaction begins, an entry is made in a rollback segment. The entry is not deleted until the transaction commits or rolls back.

The space in a rollback segment allocated for transaction information is fixed. The expected number of total concurrent transactions is a function of the number of database users and volume of activity. Thus, the total number of rollback segments should be able to accommodate the expected maximum number of simultaneous transactions. A good estimate to use is:

```
# segments = 2 x [ max # transactions / (block size) / 32 ]
```

Or you can use the following rules of thumb to determine the number of rollback segments:

<i>Number of Concurrent Transactions</i>	<i>Recommended Number of Rollback Segments</i>
less than 16	4 segments
16-32	8 segments
32 or more	n/4 but not more than 50

Setting Storage Allocations for Rollback Segments

Like data and index segments, the rollback segment is allocated in groups of blocks called *extents*. When setting storage parameters for rollback segments you should:

- Make rollback segments the same size.
- Use a high MAXEXTENTS to avoid rollback segments running out of space.

For a review of how transactions write to rollback segments, refer to Chapter 4, "Contents of the Rollback Segment" and "How Segments Are Allocated to Active Transactions."

Like other segments, the rollback segment must be able to acquire enough space to contain its data. If a rollback segment tries to acquire another extent but cannot (for example, because there is not enough space in the tablespace to allocate the extent), the current transaction will fail with an error.

Long running SELECTs can increase the amount of space needed, since rollback data must be maintained for all updates made to scanned tables over the duration of the scan. If the data required to generate a read consistent block has been overwritten, the query will get the error "snapshot too old" (described in Chapter 12).

Archiving Redo Log Data in ARCHIVELOG Mode

If the redo log's mode is NOARCHIVELOG, you may not be able to recover from media failure except by restoring a complete copy of a database made while the database was offline. If your database runs in NOARCHIVELOG mode, this section does not apply.

If you wish to guard against media failure, you will have set the redo log's mode to ARCHIVELOG and you will have ongoing responsibilities relating to recovery. You must:

- perform periodic backups of the database
- archive online redo log data.

Archiving is the process of writing data from a filled online redo log file to a secondary storage device, typically tape. Archiving causes no interruption in database operation. (However, if all online redo log files fill and they have not been archived, then instance operation is suspended until the necessary archiving is accomplished.)

Online redo log files can be archived in one of two ways: *automatically* and *manually*. These methods are described in following sections; more detail on the two alternatives can be found in the *Installation and User's Guides* since some of this information is specific to each operating system.

Log Sequence Numbers Whether you use automatic or manual archiving, you are responsible for keeping an inventory of the archived files. Every redo log file can be identified by a unique *log sequence number*. This number is assigned at the time of archive and is required at the time of recovery. Log sequence numbers begin with 1 and increment by 1 every time a new redo log file is opened.

For recovery purposes, you should archive and be able to retrieve every archived redo log file based on its log sequence number. Typically the online log files are named with a combination of the archive log destination name and the sequence number. See Appendix B and your *Installation and User's Guide* for additional details on the name and location of your log files.

You can see the status and log sequence number of current online log files using the SQL*DBA command ARCHIVE LOG LIST.

Automatic Archiving

Available for some operating systems, automatic archiving uses the background process ARCH to automatically archive online redo log files. Files can be archived to disk or directly to tape. This method may be useful for sites with system operators who are available to mount tapes whenever ARCH needs a new tape to archive to.

Automatic archiving can be enabled in two ways: using INIT.ORA and using SQL*DBA. A record of every automatic archival is written in a trace file written by the ARCH process. Each entry shows the time that archival started and stopped.

Even if you have enabled automatic archiving, you can always manually archive a redo log file.

Enabling Automatic Archiving via INIT.ORA

To use INIT.ORA to enable automatic archiving, you must adjust two parameters. Set the parameter LOG_ARCHIVE_START to TRUE and set the parameter LOG_ARCHIVE_DEST to the destination for offline logs. The specification of the destination is operating system specific; refer to the *Installation and User's Guide* for your operating system for details.

After setting the two parameters, automatic archiving will be in effect the next time the instance is started. These parameters always determine the archiving options in effect when an instance is started, although the ARCHIVE LOG command can be used to temporarily override their settings.

Enabling Automatic Archiving via SQL*DBA

You can also use a SQL*DBA command to enable automatic archiving:

```
SQLDBA> ARCHIVE LOG START [destination]
```

This will not necessarily cause an online log file to be archived immediately; rather it enables automatic archiving, so the next time an online redo log file needs archiving, it will be done automatically.

You can disable automatic archiving by using the STOP option rather than START. Regardless of the last version of this command you use, archiving is automatic each time you start an instance if the value for the INIT.ORA parameter LOG_ARCHIVE_START is TRUE.

Setting the Archive Destination

Every time a redo log file is archived, the background process ARCH must know where it should create the copy. This location is called the *archive destination*. The destination can be specified via the INIT.ORA parameter LOG_ARCHIVE_DEST or as an option to the ARCHIVE LOG command. The destination can be either a disk file or tape. As the options and specification are operating system dependent you should refer to the *Installation and User's Guide* for more details.

Disabling Automatic Archiving

You can disable automatic archiving in two ways. To temporarily stop automatic archiving, it is easier to use the SQL*DBA command:

```
SQLDBA> ARCHIVE LOG STOP
```

To disable automatic archiving as the default mode for an instance on startup, change the value for the INIT.ORA parameter LOG_ARCHIVE_START to FALSE. You must restart the instance for the new value to take effect.

When you next restart the instance, archiving will be automatic or manual as determined by the current value for LOG_ARCHIVE_START.

An error in automatic archiving will also automatically disable archiving as though the command ARCHIVE LOG STOP were used. The error may be sent to the operator's console, depending on the operating system, and it will also appear in the trace file written by the background process ARCH. After resolving the problem, you can simply resume archiving with the command ARCHIVE LOG START.

Manual Archiving

Manual archiving requires the DBA or system operator to archive online redo log files manually using SQL*DBA commands. Though this method allows you more control over when archiving is performed, it may require an explicit action (such as mounting a tape) whenever an archive is desired.

Manual archiving is the default (the INIT.ORA parameter LOG_ARCHIVE_START is initially set to FALSE).

To manually archive, you must first identify which files require archiving. You can do so using the LIST option of ARCHIVE LOG.

Then, to archive an online redo log file use the following syntax:

```
SQLDBA> ARCHIVE LOG [ NEXT | n | ALL ] [destination]
```

This will cause archival of the specified file(s) to occur immediately. For a full syntax description of this command, refer to Appendix B.

Performing Backups

To recover from media failure you must have either:

- image backups of the portion of the database to be recovered, *and* redo logs (online and offline) dating back to when those backups were taken, *or*
- a consistent backup of all database files (including redo log files and control files).

An *image backup* is a block-by-block copy of a file, in contrast to an Export, which is a logical backup, just recording the data without regard to its physical location. Image backups are useful for two purposes:

- to restore the database to a past point in time
- to restore the database to a recent or current point in time.

In the first case, the backup must consist of a consistent backup of database, log, and control files taken when the database was closed. This backup is used to restore the status of the entire database as it was at a past point in time. No media recovery is performed; the database files are simply restored.

In the second case, the backup of the database (either portions of or the entire database) would typically be used after a media failure. The backup can be of a portion of the database taken while the database was open and in use. Restoring the backup is the first step of media recovery. After the restoration, the redo log files are applied to bring the database to its status prior to the failure.

Image backups:

- should be taken regularly and frequently
- may be taken when the database is open or closed
- may be taken on a tablespace basis, rather than backing up the entire database.

More frequent backups will minimize the time potentially required for media recovery. Backups on individual tablespaces gives you added flexibility and control over media recovery, as all tablespaces of a database need not be restored from the same backup point.

Backup and recovery procedures vary by operating system and are discussed in more detail in the *Installation and User's Guides*.

Backing Up a Closed Database

When the database is closed, you can backup the entire database. This type of backup is called a *consistent backup*. You can then use the backup to return the database to that state.

To back up a closed database:

1. Shutdown the instance.
2. Use your preferred operating system backup utility to copy all database files, online log files, and control files.
3. Restart the instance if desired.

Backing Up an Open Database

If you use ARCHIVELOG mode, you may back up all or part of your database while it is running and being accessed; this is called a *hot backup*. This will allow you to restore those tablespaces in the event of media failure.

Backups that are taken while the database is open have the following characteristics:

- The backup allows users to have normal access (including update) to all online tablespaces during the backup. Thus, users can access the tablespace being backed up.
- When used for recovery, the backup can only be used to return to the most recent state of a database, not to a previous state.
- Only the database files comprising a tablespace are backed up; the log files and control files are not backed up.

To back up a tablespace in an open database use the following steps. To back up an entire open database, repeat these steps for every tablespace in the database. You can back up the SYSTEM tablespace using this procedure.

1. You must be running in ARCHIVELOG mode and archiving the redo log files as they fill.
2. If the tablespaces containing the files to be backed up are online, you should enter the SQL statement:

```
SQLDBA> ALTER TABLESPACE tablespace BEGIN BACKUP
```

This statement is not necessary if the database is closed, or if the tablespaces are offline.

3. Use your preferred operating system backup utility to copy *all* files associated with each tablespace. See your *Installation and User's Guide* for more information on the preferred backup command. The backup may reflect some of the updates occurring during the

backup, depending on where in the physical file the updates are written (if they are written before the backup reaches that portion of the file).

4. Enter the SQL statement:

```
SQLDBA> ALTER TABLESPACE tablespace END BACKUP
```

Do not forget to use both ALTER commands. Though the commands appear to have little effect on users (who can still access the tablespace normally), the RDBMS uses the commands to identify the portion of the online redo log that was created during the time of the backup.

If you forget to enter ALTER TABLESPACE ... BEGIN BACKUP and the subsequent shutdown is not normal, then the tablespace's files may be backed up incorrectly and you will not be able to use these files for recovery.

If you forget to enter ALTER TABLESPACE ... END BACKUP and the system continues to run, then at the next instance startup, the ORACLE RDBMS will assume that media recovery is necessary, and may require the use of offline log.

Media Recovery Procedures

The following sections describe media recovery procedures when different ORACLE files are inaccessible (either cannot be written to or read).

Recovering database files to a current state requires that you archive online redo log files as they fill (the redo log is used in ARCHIVELOG mode).

Note: It is your responsibility to archive log files and to be able to locate each file based on its identifier, the log sequence number.

You can recover either a database or a single tablespace. You may restore a single database file in a tablespace in order to recover the entire tablespace. The process of tablespace recovery requires that all files in a tablespace be taken offline temporarily.

Recovering a Database File not in the Tablespace SYSTEM

If ORACLE discovers it cannot **read** a database file in other than the SYSTEM tablespace, an operating system error will be returned, but the database system will continue to run, with the error being returned each time an unsuccessful read occurs.

If ORACLE discovers it cannot **write** to a database file in other than the SYSTEM tablespace, an error occurs and the tablespace should be taken offline.

If media failure occurs in a database file in other than the tablespace SYSTEM you may:

- recover just the affected *tablespace*, leaving the rest of the database open and running for normal use during recovery. In this case, you must recover to the *present*.
- shutdown the instance and restore or recover the *entire database*. In this case, users cannot access the database during recovery and you can recover to an arbitrary point in time (see "Point in Time Recovery" later in this chapter).

Tablespace Recovery

To recover a file in a tablespace *other than the SYSTEM tablespace*, follow these steps. You will need access to two operating system terminal (interactive) sessions (one to initiate and perform recovery, and one to copy offline log files to online storage during recovery).

1. The instance should be started and the database open. Users may be accessing the system, or you may startup the instance in DBA only mode.

2. From one terminal session with SQL*DBA access, connect with a DBA username.
3. Take the tablespace with the problem file offline:

```
SQLDBA> ALTER TABLESPACE tablespace OFFLINE
```
4. Use your preferred operating system commands to restore the most recent backup copies of each database file you wish to recover. You need only restore copies of files with known access problems, even if this means you restore one out of many files in a tablespace. (You can restore additional files if you want.) The backup copies need not have been taken at the same point in time. Do not restore any control files.
5. Enter the SQL*DBA command (see Appendix B for the full syntax description):

```
SQLDBA> RECOVER TABLESPACE tablespace [, tablespace ...]
```
6. If no offline redo log files are required to complete the recovery (because enough information is available in the online redo log files to recover the file) you will soon see a message stating that recovery is complete.
If offline redo log files are required, SQL*DBA will identify the log sequence numbers of the offline redo log files as it needs them. From the other terminal session, you may use the operating system file copy facilities to copy the offline log file from tape to disk. Where you copy the files is not important; SQL*DBA will prompt for the name of the file in its current directory or disk location. The name is unimportant so long as ORACLE can open the named file. ORACLE checks that the file is the expected one by validating its log sequence number.
Alternatively, you can restore directly from tape, if the tape is mounted and you reply with the tape specification when prompted for the next file's location.
SQL*DBA continues prompting for offline redo log files as it needs them, re-applying their contents.
If you know which offline redo log files will be needed, you can avoid delays between the prompts or using two terminal sessions. Simply copy them from tape to disk ahead of time. The first redo log file you will need is the one being written at the time of the oldest backup database file you restored. The last redo log file you will need is the one being written at the most recent time the tablespace was taken offline.

Database Recovery

7. Bring the tablespace back online:

```
SQLDBA> ALTER TABLESPACE tablespace ONLINE
```

You can also recover a database file in a tablespace other than SYSTEM by recovering the entire database. The steps are similar to the previous steps, however, users cannot access the database during the recovery process.

1. The instance should be started and the database mounted exclusively but not open.
2. Restore backup copies of each database file you wish to recover. You need only restore copies of files with known access problems, even if this means you restore one out of three files in a database. The backup copies need not have been taken at the same point in time. Do not restore copies of the control files.

3. Enter SQL*DBA and enter the command:

```
SQLDBA> RECOVER DATABASE
```

4. If no offline redo log files are needed to complete the recovery (enough information is available in the online redo log files to recover the file) you will soon see a message stating that recovery is complete.
If offline log files are needed, see Step 6 in "Tablespace Recovery".
5. After the recovery complete message displays, the database can be OPENed, if you wish to open it exclusively. To allow other instances to also access the database you can DISMOUNT, SHUTDOWN, STARTUP, MOUNT SHARED, and OPEN the database.

Interrupting Recovery

You can interrupt recovery at any time:

- if at a prompt for a log file name, enter the word CANCEL
- if in the middle of recovering a file, use your operating system's interrupt signal.

Resuming Recovery

To resume recovery, just reenter:

```
SQLDBA> RECOVER TABLESPACE tablespace [,tablespace ...]  
or:  
SQLDBA> RECOVER DATABASE
```

Recovery will resume at the file where it left off; it will not begin again. If you want to restart recovery at an earlier file or backup copy, you should restart the recovery process.

Recovering a Database File in the Tablespace SYSTEM

If you cannot read or write a database file belonging to the SYSTEM tablespace, then the database cannot be accessed during recovery. The SYSTEM tablespace is required to open a database, and the SYSTEM tablespace cannot be available when it is being recovered.

In this situation, you should follow the five steps for database recovery (not tablespace recovery) described previously.

Recovering an Online Redo Log File

If an online redo log file suffers media failure, the database system will stop with an operating system read/write error. If you know the cause of the problem and you are positive that data has not been lost, then you can simply restart the instance and ORACLE will perform instance recovery.

If you suspect that redo log data may have been lost, then you have the following choices:

- Restore the database using a consistent set of backup files (including the database files, online log files, and control files). A consistent set of backup files means that all files were written at the same time while the database was shutdown. After recovery the database will reflect its state as of that backup.
- Restore the database as described in "Database Recovery" and in the file RECOVERY.DOC to recover only through the last readable redo log file. Thus, only changes committed to that point are restored; changes made after that date are lost. The point is an arbitrary point before the failure occurred in the last readable redo log. See "Point in Time Recovery" later in this chapter.
- Create a new database and perform a full database import, using any cumulative and incremental export files which followed the last full database export. This will restore the database to the point of the most recent incremental export.

In some operating systems you can reduce the risk of losing online log files by using the operating system's online duplexing facility. If a realtime copy of the online log is kept on a separate disk drive than the one written directly, then you are more likely to have one good copy in the event of media failure affecting the redo log.

Recovering Control Files

Because control files are required by an instance in order to start or recover a database, they require guaranteed protection against failure. In general, the writing, protection, and recovery of these files is automatic, requiring little DBA action other than specifying multiple control files on different disks.

Symptoms of control file failure include the failure of an instance to start.

If not all control files are damaged, simply copy a good control file. The recommended way to copy a control file follows:

1. Shutdown the instance.
2. Copy the good control file using a new name. To be conservative, do not use the name of the damaged file, at the risk of copying the wrong file or overwriting a good file.
3. Edit the INIT.ORA file value for the parameter CONTROL_FILES. Delete the name of the old, damaged file and add the filename of the new copy.
4. Restart the instance.

If all control files are damaged, then the entire database must be restored, using one of the following methods:

- Restore the database using a full consistent backup (backup copies of database, online log, and control files all taken at the same time while the database was closed). This also restores the database to a point in the past, with backup copies of the control files.
- Perform media recovery for the entire database, using a backup copy of the control file. See the file RECOVERY.DOC. This restores the database to a point in the past, with backup copies of the control files.
- Create a new database and import the last full database export file taken, along with any associated cumulative or incremental export files. This restores the database to the point of the most recent export and generates new control files.

Point in Time Recovery Normal recovery usually means recovering the entire database to the present point in time, as documented in this chapter. In unusual circumstances you may need to recover to a past point in time. Refer to the file on your distribution media, called RECOVERY.DOC, for instructions and restrictions pertaining to point in time recovery. This file describes steps for recovering to the end of a particular log file or to a specified point in time.

Note: If you anticipate needing this type of recovery, you should also take consistent image backups of your control file (as described in RECOVERY.DOC) before and after changing the file configuration (as when altering tablespaces to add files, or when dropping tablespaces).

Export/Import

The Export/Import utility allows you to archive and retrieve ORACLE data, using operating system files. While these utilities may be used for recovery, their main purposes are somewhat different, and include:

- saving snapshots of table definitions or data at some point in time (historical archive)
- saving table definitions (with or without the data) offline with the intention of reloading them into an ORACLE database at a later time
- moving ORACLE data from a database on one CPU to a database on another CPU
- moving ORACLE data between two versions of ORACLE software.

However, Export and Import have some limitations:

- A database must be running in order to perform export or import.
- Export files cannot be edited and can be used only by Import.
- Import has limited flexibility in dealing with export files and must import the data as is (for example, it cannot perform a conditional import).
- Exported data is only a logical representation of the data, and can be used for recovery from media failure only to the time of the most recent export.

The following sections describe the use of Export and Import relative to backup and recovery procedures. Refer to the *ORACLE Utilities User's Guide* for more complete information about using Export and Import.

Export

Export is a utility to move database data to an operating system file. DBAs can export all data in a database, or specify which data should be exported. For example, you can specify:

- only tables whose data has been updated since the last export (via cumulative or incremental exports)
- which types of database objects are to be exported (including tables and data, table definitions only, grants, views, clusters, indexes)
- individual tables
- all tables for one user.

Thus, you have much flexibility in choosing what data to export. The data is copied in non-editable format to an operating system file; data in the database is unchanged.

Export must be used while the database is running. Users may concurrently access tables being exported. Each single table's data will be consistent as export uses a read consistent view of a table's data.

You may, however, want to export a consistent view of all tables used by an application. In this case the export should be taken when users are not accessing the tables, so that all data across tables is in a steady state. The easiest way to do this is to use the SQL statement LOCK TABLE table in SHARE MODE on all tables used by the application before running the export. Or, you could disable the application either by temporarily revoking privileges to the tables or by running the database in DBA-only mode, rather than by shutting down ORACLE.

Full Database Exports

Regularly performed full database exports are an important part of a thorough backup strategy. The purpose of a full database export is to save all table data and definitions in the database as of a certain point in time, so that the database or individual tables can be reconstructed in the future, if necessary or desired.

A database export differs from a database image backup or disk copy. The export saves the data and database definitions in a format usable only by ORACLE. It is a *logical* copy of ORACLE data, and when restored by ORACLE the data might be located in a totally different area on disk. In contrast, an image backup is a *physical* copy, recording the location of all data in order to restore the data to the same form.

One way to restore a database after media failure is to import one or several export files. For example, you might recreate a database, import one full database export file, one cumulative export file, and two incremental export files.

In addition to full database exports, you can use two other types of export: *incremental* and *cumulative*. An incremental or cumulative export identifies and exports tables whose data has changed in a given period of time. These exports typically require less time and storage to complete. They can be useful for recovering a table that was erroneously dropped, for example, while keeping the rest of the database in its current state.

Incremental Exports	<p>An incremental export exports only objects which have been updated since the last export of any type (full, incremental, or cumulative). Incremental exports are typically taken more often than cumulative exports or full database exports.</p>
Cumulative Exports	<p>A cumulative export backs up tables that have changed since the last cumulative or full database export. Thus, it is similar to an incremental export but with longer intervening periods of time. (If incremental and cumulative exports were both taken with the same frequency, they would be identical exports.)</p> <p>A cumulative export can be used to compress a number of incremental export files into a smaller number of cumulative export files; it is not necessary to save incremental export files taken before a cumulative export.</p>
Import	<p>Import is the complementary utility to Export. Import reads files created by Export and writes the data back into a database. When importing a file, you have some flexibility about which data to import from the file.</p>
Using IMPORT for Full Database Recovery	<p>Some circumstances may call for the recreation of a database, due to the loss of files required for operating a database. One example is the loss of all control files, although in this case there are other alternatives as well.</p> <p>If you decide that a full database import is the best solution for your situation, follow these steps:</p> <ol style="list-style-type: none"> 1. Locate all export files containing data you wish to import. You may use one file containing all database data, or several export files of different types (full database, incremental, cumulative, or individual user). You can import directly from tape. 2. Close the database. 3. Shutdown all instances accessing it. 4. Edit the INIT.ORA file if necessary. 5. Follow the database creation steps in Chapter 13, Initial Database Creation. When database creation is complete, the database will be left open. 6. Perform an import using the desired export file(s).

16

SPACE MANAGEMENT

Everyone is a genius at least once a year. The real geniuses simply have their bright ideas closer together.

George C. Lichtenberg

Unused capacities atrophy, cease to be.

Tillie Olsen

This chapter describes managing the database structures and altering them as necessary to improve storage or performance. Contents include:

- adding, altering, renaming, dropping rollback segments
- adding, enlarging, renaming, dropping tablespaces
- calculating space required by tables, indexes, and clusters
- monitoring space used by temporary segments.

Portions of this chapter are of interest primarily to DBAs — discussions of managing the operating system files (database, redo log, and control files). Discussions of managing logical database structures, such as tablespaces, tables and clusters, indexes, and temporary segments, may also interest database users and application designers.

The following chapters also contain information relating to this chapter:

- Chapter 3 ORACLE RDBMS File Structure
- Chapter 4 Tablespaces and Segments
- Chapter 5 User Database Objects

Database File Maintenance

Physical database files map to logical divisions of a database called tablespaces. A tablespace must have at least one file; it may have many files. Refer to Chapter 3 for descriptions of the database files.

Because of the close relationship between files and tablespaces, most file maintenance is done using SQL statements referencing tablespaces. For example, using CREATE TABLESPACE or ALTER TABLESPACE you can:

- specify the original database file(s) to map to a tablespace
- add a file to a tablespace
- rename a file in a tablespace (using either ALTER TABLESPACE or ALTER DATABASE).

Database files cannot be dropped or enlarged. Rather than enlarging a database file, simply add another file to one of the tablespaces.

For additional information on database file maintenance, see the sections on tablespace maintenance later in this chapter.

Redo Log File Maintenance

The redo log files are used to record changes made to database blocks when transactions are committed and for rolling forward during recovery. These files are described in Chapter 3; for a description of using the redo log for recovery, refer to Chapter 15.

Redo log maintenance options depend somewhat on how you are using the redo log. As described in Chapter 3, the redo log can be used in two different ways: ARCHIVELOG mode and NOARCHIVELOG mode.

The following sections describe altering the redo log files.

Note: To perform any maintenance on the redo log files two conditions are required:

1. The database must be mounted but closed.
2. Connect to the database with the keyword INTERNAL

Then, use the ALTER DATABASE command to add, drop, or rename log files. After completing your database alteration, shutdown the instance and then restart.

Adding New Redo Log Files

You may want to add new redo log files for the following reasons:

- increase online log storage
- gain more flexibility regarding when you archive the online redo log files (only if you are using ARCHIVELOG mode and especially if using manual archiving)
- change the size of an online file (by adding larger or smaller files, and dropping current redo log files).

You gain flexibility by adding files, because then you need not archive a file as soon as it fills. With two online redo log files, you must archive the first because when the second file fills, the LGWR background process must immediately switch back to the first. However, if you have 3 or 4 files, you can batch 2 or 3 files together to archive at one time.

After connecting as described above, you can add a new log file with the following syntax:

```
ALTER DATABASE name  
ADD LOGFILE filespec [ , filespec ]
```

All redo log files should be a minimum of 50 Kbytes in size.

You can add new redo log files until you hit the lower of the limits on log files: LOG_FILES in INIT.ORA or MAXLOGFILES from CREATE DATABASE. After completing your database alteration, shutdown the instance and then restart.

Dropping Redo Log Files

You may wish to drop online redo log files in order to change the size of your log files or to decrease the number of redo log files that the database may write to.

To drop a redo log file, connect as described above, and then use the following syntax:

```
ALTER DATABASE name  
DROP LOGFILE filespec [ , filespec ]
```

Do not drop redo log files until you are sure:

- they are not being currently written
- they have already been archived (if you are using ARCHIVELOG mode).

After completing your database alteration, shutdown the instance and then restart.

Renaming Redo Log Files

You may wish to rename redo log files in order to specify a file which is larger, smaller, or in a different location than existing redo log files.

After connecting as described above, you can rename a redo log file with the following syntax:

```
ALTER DATABASE name  
RENAME FILE filespec [ , filespec ] TO  
filespec [ , filespec ]
```

(This statement can also be used to rename database files.) A file by the new name must exist at the time you execute this statement. The filename should be new and not a duplicate of an existing file, or results will be unpredictable. After completing your database alteration, shutdown the instance and then restart.

The execution of this command affects only information stored internally in the database; it is not equivalent to using an operating system command to rename a file.

See your *Installation and User's Guide* for more details on adding files or tablespaces.

Rollback Segment Maintenance

As described in Chapter 4, rollback segments contain information required for read consistency and to undo changes made by transactions that rollback. Most databases will have multiple rollback segments.

The original rollback segment, called SYSTEM, is automatically created as a part of CREATE DATABASE. It is created in the SYSTEM tablespace and its initial size is defined by the default storage parameters used by the SYSTEM tablespace.

If there are multiple tablespaces, there must be at least one rollback segment in addition to the SYSTEM rollback segment. A shared disk system also requires at least one rollback segment per instance accessing the database, in addition to the SYSTEM rollback segment.

Creating New Rollback Segments

A DBA must use the following SQL syntax to create new rollback segments:

```
CREATE [ PUBLIC ] ROLLBACK SEGMENT name  
[ TABLESPACE tablespace ]  
STORAGE ( [ INITIAL n ] [ NEXT n ]  
[ MINEXTENTS n ] [ MAXEXTENTS { n | NULL } ]  
[ PCTINCREASE n ] )
```

A rollback segment is not in use unless it is acquired at instance startup; thus a rollback segment that has just been created will not be used until an instance has shut down and re-started, and acquired the segment either as a public segment or through naming it in its INIT.ORA file.

You may create new rollback segments in any tablespace, but it is best to create rollback segments in tablespaces which will remain online; a tablespace cannot be taken offline if it contains an active rollback segment.

Public versus Private Rollback Segments

Public rollback segments form a pool of segments that can be acquired by any instance needing an additional rollback segment, while private rollback segments are usually associated with one and only one instance. By default a rollback segment is private (with the exception of the SYSTEM rollback segment) and can be used only by an instance naming it in ROLLBACK_SEGMENTS.

To create a public rollback segment, create it with the keyword PUBLIC. Public segments are typically not specified in INIT.ORA for any particular instance because they are assumed to be available to any

instance needing them. However, you can name a public segment in INIT.ORA for the parameter ROLLBACK_SEGMENTS.

To create a private rollback segment, simply do not use the keyword PUBLIC when creating it. Private segments will not be used unless they are named as the value for ROLLBACK_SEGMENTS in the INIT.ORA file for one instance.

Monitoring Rollback Segments

To monitor how much space rollback segments are using, query the data dictionary views DBA_ROLLBACK_SEGS and DBA_SEGMENTS. You can also use the SQL*DBA command MONITOR ROLLBACK to see information about rollback segments.

Changing the Size of Rollback Segments

You can increase or decrease the storage parameters for a rollback segment with the SQL statement:

```
ALTER [ PUBLIC ] ROLLBACK SEGMENT name  
    STORAGE ( [ NEXT n ]  
        [ MAXEXTENTS { n | NULL } ]  
        [ PCTINCREASE n ] )
```

This command only affects future extents used by the rollback segment; existing extents are not affected. If the rollback segment is a public one, you must use the keyword PUBLIC. This command is not valid for changing the private/public status of a rollback segment.

Dropping Rollback Segments

You may wish to drop a rollback segment for reasons such as:

- a segment becomes too fragmented, having many extents
- a segment is no longer needed because the instance using it is no longer accessing the database
- you want to recreate it as PUBLIC or private
- you want to locate it in a different tablespace.

To drop a rollback segment follow these steps:

1. Check that the database is open.
2. Connect using a DBA username.
3. Make sure the rollback segment to be dropped is not in use by any instance, by querying the data dictionary view DBA_ROLLBACK_SEGS.

4. Enter the SQL statement:

```
DROP [ PUBLIC ] ROLLBACK SEGMENT name
```

If a rollback segment is required to restart the system, create another one.

Tablespace Maintenance

As described in Chapter 4, the tablespace is the primary logical division of a database. It is the unit used for space allocation, online availability, and recovery. The following sections describe the following tablespace maintenance functions:

- creating new tablespaces
- monitoring and controlling space usage within a tablespace
- increasing the size of a tablespace (by adding a file)
- setting or changing default storage parameters associated with the tablespace
- renaming files in a tablespace
- taking tablespaces offline and online
- dropping tablespaces.

The related SQL statements (CREATE TABLESPACE, ALTER TABLESPACE, and DROP TABLESPACE) require DBA privilege. Some of the options for these statements may only be executed while a tablespace is offline.

Creating a New Tablespace

To make more room for data, you can enlarge a database in two ways:

- add a new tablespace using CREATE TABLESPACE
- add a new file to an existing tablespace, using ALTER TABLESPACE, to make that tablespace larger.

You may wish to add a new tablespace for any of the following reasons:

- for the specific allocation of temporary segments
- to spread and control I/O contention
- to facilitate your backup/recovery scheme
- to control resource use on a user basis
- to separate table and index storage.

To create a new tablespace use the following syntax:

```
CREATE TABLESPACE tablespace
DATAFILE filespec [ , filespec ] ...
[ DEFAULT STORAGE (
    [ INITIAL n ] [ NEXT n ]
    [ MINEXTENTS n ] [ MAXEXTENTS { n | NULL } ]
    [ PCTINCREASE n ] ) ]
[ ONLINE | OFFLINE ]
```

At least one operating system file must be specified to correspond to the tablespace. The specification of *filespec* varies by operating system.

By default, a tablespace is online after it has been created.

Note: If this is the first tablespace to be added after SYSTEM, and you have only one rollback segment, then you must also create one additional rollback segment to accommodate the new tablespace. You must create the second rollback segment *before* creating the second tablespace.

Enlarging a Tablespace (Adding a File)

To increase the size of a tablespace, you must add a new file to it. This will always be a subsequent file for a tablespace, as the first file is specified in the CREATE TABLESPACE command. To add a new database file, follow these steps:

1. The database should be open; the tablespace can be online or offline.
2. Enter SQL*DBA and connect to the database as a DBA.
4. Use the following syntax to add the file:

```
ALTER TABLESPACE tablespace
ADD DATAFILE filespec [ , filespec ] ...
```

The syntax for *filespec* is operating system dependent; see your *Installation and User's Guide* for details. Be sure you use a new filename; if you use the name of an existing database file, results are not predictable.

Monitoring Space Usage in a Tablespace

Several data dictionary views allow you to monitor how much space is allocated, used, or free in a tablespace. In particular, you may find the following views useful:

- DBA_SEGMENTS
- DBA_EXTENTS
- DBA_FREE_SPACE

Setting User Quotas for Tablespace Usage

When you grant a user RESOURCE privilege to a tablespace, you may optionally specify a limit on how many bytes that user may use in that tablespace. For example:

```
GRANT RESOURCE (1300) ON TSPACE_TWO TO NANCY
```

The same statement can be used to impose or alter a quota. If the user has already used more space than the new quota allows, then the user cannot consume more space until his storage drops below the new limit.

Setting Default Storage Parameters for a Tablespace

One characteristic of a tablespace is that it sets the default storage parameters and limits for tables or indexes created in the tablespace. When creating the tablespace, you can set values for the storage parameters using the storage options for the CREATE TABLESPACE statement.

Refer to Chapter 4 for a description of the storage parameters, such as INITIAL, NEXT, and PCTINCREASE.

Changing Default Storage Parameters

To alter the default storage parameters for a tablespace, use the following SQL statement (note the keywords DEFAULT STORAGE):

```
ALTER TABLESPACE tablespace
DEFAULT STORAGE (
  [ INITIAL bytes ]  [ NEXT bytes ]
  [ MINEXTENTS int ] [ MAXEXTENTS int ]
  [ PCTINCREASE pct ] )
```

Renaming Files in a Tablespace

You may wish to rename files in a tablespace in order to relocate files to different devices or to use files of different sizes.

To rename database files use the following SQL statement:

```
ALTER TABLESPACE tablespace
RENAME DATAFILE text [ , text ] ... TO text [ , text ] ...
```

Taking a Tablespace Offline

You may wish to take a tablespace offline for any of the following reasons:

- to make the data in the tablespace (temporarily) unavailable
- to free the disk space used by the tablespace for another purpose
- to perform tablespace maintenance (such as adding a file or performing backup) though a tablespace need not be offline to do so.

To take a tablespace offline follow these steps:

1. Check that the database is open.
2. Connect to the database using the keyword INTERNAL.
3. Verify that there are no active rollback segments in the tablespace, by querying the data dictionary view DBA_ROLLBACK_SEGS. If there are active segments, you cannot take the tablespace offline.
4. Use the following syntax:

```
ALTER TABLESPACE tablespace  
OFFLINE [ NORMAL | IMMEDIATE ]
```

Use NORMAL to indicate that the tablespace should only go offline when all users are finished using it, or use IMMEDIATE to indicate that it should go offline in any case.

A tablespace will remain offline (through instance startup and shutdown) until you bring it back online.

If a tablespace contains indexes for tables which reside in other tablespaces, queries against those tables may or may not execute, depending upon the access path chosen by the optimizer. For more explanation of which statements may succeed, refer to Chapter 4.

Bringing a Tablespace Online

In order to access data within a tablespace it must be online. Follow these steps to bring an offline tablespace online:

1. Check that the database is mounted but shut.
2. Connect using the keyword INTERNAL.
3. Enter the SQL statement:

```
ALTER TABLESPACE tablespace ONLINE
```

Dropping Tablespaces

Tablespaces may be dropped, even if they have table data in them. To drop an online tablespace use the syntax:

```
DROP TABLESPACE tablespace [ INCLUDING CONTENTS ]
```

If a tablespace containing data is to be dropped, the option INCLUDING CONTENTS must be used for it to be dropped successfully.

Managing Storage for Tables and Clusters

Owners of database tables and clusters have several options relating to data storage, including:

- in which tablespace the table or cluster should reside
- how much space should initially be allocated
- how much space should be allocated each time more space is required
- how empty and\or how full data blocks may become, on average
- how much total space a table or cluster may use.

Storage is set using the SQL statements:

- CREATE TABLE or CREATE CLUSTER
- ALTER TABLE or ALTER CLUSTER
- CREATE TABLESPACE (to set default storage parameters for the tablespace containing the table or cluster).

Any parameter specified in CREATE TABLE will override the same parameter in the CREATE TABLESPACE statement. Thus, a table may "inherit" storage parameters from its tablespace if, when the table was created, its CREATE TABLE statement did not specify every storage parameter.

Cluster storage parameters override storage parameters specified for individual tables in the cluster.

Locating Tables in Tablespaces

You can create a table in any tablespace for which you have sufficient RESOURCE privilege by using the TABLESPACE option for CREATE TABLE. If no tablespace is specified, then a table is created in the default tablespace assigned to a user with the ALTER USER command, or the SYSTEM tablespace if no other tablespace has been assigned.

Using multiple tablespaces, a DBA can spread storage and I/O across tablespaces and devices. He can do this through:

- assigning default tablespaces to users
- granting of RESOURCE privileges on the tablespaces
- giving users space quotas in tablespaces where they have RESOURCE.

Setting Table Storage Parameters

Some storage parameters can be set in both the CREATE TABLE and the CREATE TABLESPACE statements. These parameters, described in Chapter 4, are:

INITIAL
NEXT
MAXEXTENTS
MINEXTENTS
PCTINCREASE

Storage parameters unique to the CREATE TABLE and CREATE CLUSTER statements and described in Chapter 5 are:

PCTFREE
PCTUSED

Calculating Space Required by a Table

To estimate the space required by a single, non-clustered table, you can use the following formula. The formula takes the following items into consideration:

- x is the number of rows per table.
- y is the number of columns per table.
- len is the average column length.
- A block requires an average of 90 bytes of overhead.
- A row requires an average of 5 bytes of overhead, including information in the block's row directory.
- A column requires 1 byte of overhead.

The fully annotated formula resulting in the *number of blocks* required per non-clustered table is:

$$\# \text{ blocks} = \frac{x \text{ rows} * (5 \text{ bytes} + y \text{ cols} * (1 \text{ byte} + len))}{(\text{blocksize} - 90 \text{ bytes}) * (1 - PCTFREE/100)}$$

or expressed without units and a bit more concisely:

$$\# \text{ blocks} = \frac{x * (5 + y * (1 + len))}{(\text{blocksize} - 90) * (1 - PCTFREE/100)}$$

To calculate the *number of bytes*, multiply the result of the above calculation by the blocksize in bytes. The total number of bytes can be used for the INITIAL storage parameter. Less space will be required if you have a large number of trailing nulls, and more space if you have a large number of long columns or rows.

Note that average column length may have a wide deviation, depending on the nature of the table's column definitions and actual data.

Monitoring Actual Space Used

To see how much space is used by rows of a table you can use a query something like:

```
SELECT COUNT(*) FROM table
GROUP BY SUBSTR (ROWID, 6, 14)
```

This will give you the number of rows per block, which you can multiply time the average length of a row, to get the average amount of space required per block.

first 2 bytes block

Rowid 6,13 → less points sent completed

Increasing Table Storage Parameters

Once a table exists, its storage parameters can be changed, using either of the commands ALTER TABLE or ALTER TABLESPACE. All parameters except INITIAL and MINEXTENTS can be altered.

No structures currently allocated for a table are changed; thus, existing extents will not change in size. If PCTFREE and PCTUSED are altered, the new algorithm for using space will apply to existing blocks.

Managing Index Storage

Indexes for tables are created and stored independently from the tables. An index can be created and dropped anytime after a table is created. To see the format of an index block, refer to Chapter 5.

Locating Indexes in Tables espaces

Indexes can be created in any tablespace where the creator has RESOURCE privilege. Thus, all the indexes for a table can be in different tablespaces (and thus on different devices) from each other and from the table's data.

To guarantee access to a table, all tablespaces containing its data and indexes must be online.

Calculating Space Required by an Index

You can estimate the space required by an index that is created *after* the data has been loaded. Assume the following variables:

- x is the number of rows in the table.
- z is the number of columns indexed.
- len is the average length of the indexed columns.

- A non-unique index requires row overhead of 11 bytes and a unique index requires row overhead of 10 bytes (6 for the ROWID, 3 for row overhead, and 2 for the row directory)

To find the *number of blocks* required by the *leaf blocks* of an index, use the following calculation:

$$\# \text{ leaf blocks} = \frac{x \text{ rows} * (11 \text{ bytes} + z + \text{len})}{(\text{blocksize} - 90) * (1 - \text{PCTFREE}/100)}$$

To calculate the *number of bytes* required, multiply the result by the ORACLE blocksize.

To calculate the *number of bytes required by the entire index* (including the branch blocks), multiply the result in bytes by about 1.1. This is based on the assumption that the branch blocks require about 10% additional blocks.

Choosing Storage Parameters for Indexes

You should usually use the default storage PCTxxx parameters when creating indexes. It is best if index data is densely packed, so that fewer blocks need be read when using the index. Note that the PCTFREE value is only used when indexes are created; PCTFREE is not used when new values are added to an existing index.

For UNIQUE indexes, you should always use the default PCTFREE value.

For non-UNIQUE indexes and when you anticipate adding more rows with duplicate index values in the future, you may increase PCTFREE to reduce the potential amount of index block splitting.

The storage allocated for an index can be altered with the SQL statement ALTER INDEX. Changes in storage parameters will effect only future extents, not extents which currently exist.

Limits on Indexes

Up to 16 columns may be included in a single index. The total width of an index key may not exceed (block-size/2 - x) where x is an operating system dependent amount of overhead, varying from 30 to 60 bytes.

Managing Clusters

Clusters are an alternate method of storing data in an ORACLE database. By clustering tables, you can speed retrieval time for certain queries by locating related data in the same blocks. Refer to Chapter 5 for more details on clusters. Following sections describe controlling storage used by clusters.

Creating Clusters

The use of a cluster is particularly recommended when two or more tables are often joined in queries. See Chapter 5 for a discussion of considerations relating to clustering.

How much storage a cluster will require depends on the nature of the data and the cluster key chosen.

Choose cluster key columns carefully:

- If more than one column is common to the clustered tables, make the cluster key a concatenated key.
- Too little data stored for each key (few rows per cluster key) may waste space and earn negligible performance gains.
- Too much data per key may remove the benefit of clusters by requiring many cluster index lookups.

Single-Table Clusters

A cluster may consist of one table only. Occasionally the data of a single table will warrant clustering that table. Such tables often have one or more columns with several values in common and those columns are frequently queried.

Choosing the SIZE Parameter for Clusters

Clustered data is stored similarly to non-clustered table data. The storage parameters are the same and may be set and altered just as they are for tables. The SIZE parameter, however, is distinct to clusters.

When creating a cluster, use the SIZE option for the CREATE CLUSTER statement to indicate the average space required by all entries per cluster key. The SIZE option does not limit the number or size of entries per cluster key, but helps ORACLE RDBMS calculate on the average how many cluster keys (and rows) can be stored per ORACLE block.

The SIZE option:

- is specified in bytes
- applies to all tables in the cluster
- is better overestimated than underestimated.

Monitoring Space used by Clusters To monitor space used by clusters, query the data dictionary views DBA_CLUSTERS, DBA_USER_SEGMENTS and DBA_EXTENTS (or the USER versions of these views).

Loading Clusters After a cluster is created, you must add tables to the cluster using the CREATE TABLE statement. Variations of this statement allow you to cluster new or old tables:

To cluster a *new* table, use the following statement:

```
CREATE TABLE new_table  
(col1 datatype, col2 datatype, ..., coln datatype)  
CLUSTER cluster_name (table_column)
```

To copy an *existing* table into a cluster, first use the following statement to create, cluster, and load a new table:

```
CREATE TABLE new_table_name  
CLUSTER cluster_name (table_column)  
AS SELECT * FROM existing_table
```

Your SELECT clause may retrieve all rows or just a subset. You can load portions of the cluster at a time to reduce the requirements for rollback segment space, if necessary.

Then, after verifying that the data was copied correctly, drop the old table which was not clustered:

```
DROP TABLE existing_table
```

Then rename the table you created in a cluster to the previous table's name if you wish:

```
RENAME new_table_name TO existing_table_name
```

"Empty" or "full" tables may be clustered keeping the following guidelines in mind:

- Less overhead is entailed if empty tables are added, because the change only requires updates in the data dictionary and changing the initial blocks assigned to the table. No table data need be physically relocated in the database files.
- More processing is required to copy tables with data, because all the existing data must be read and moved from the table's blocks to the cluster's blocks, requiring many rollback segment entries.
- Bulk loading a table into a cluster from an existing table can place a heavy demand on the system.

Dropping Clusters

Before a cluster can be dropped, each table in the cluster must be dropped from the cluster unless the option INCLUDING TABLES is specified.

Dropping tables from clusters will take longer than dropping unclustered tables, and will prevent other updates on any other tables in the cluster.

Managing Temporary Segments

Though users never directly see, create, or use temporary segments, the ORACLE RDBMS often requires temporary segments to execute user statements. See Chapter 4 for a list of statements which may use temporary segments. The ORACLE RDBMS will create temporary segments on behalf of any database user, regardless of the database privileges that user may have.

This section describes the control you have over temporary segments.

Monitoring the Number of Temporary Segments

When a database is created or started, no temporary segments exist. As a database is used, temporary segments are created as needed. A number of temporary segments are created for a user during his session. Each segment is dropped when it is no longer needed.

To see how many temporary segments exist, and how much space they are using, query the data dictionary views USER_SEGMENTS, ALL_SEGMENTS, or DBA_SEGMENTS, where the value for column SEGMENT_TYPE equals TEMPORARY.

Monitoring the Location of Temporary Segments

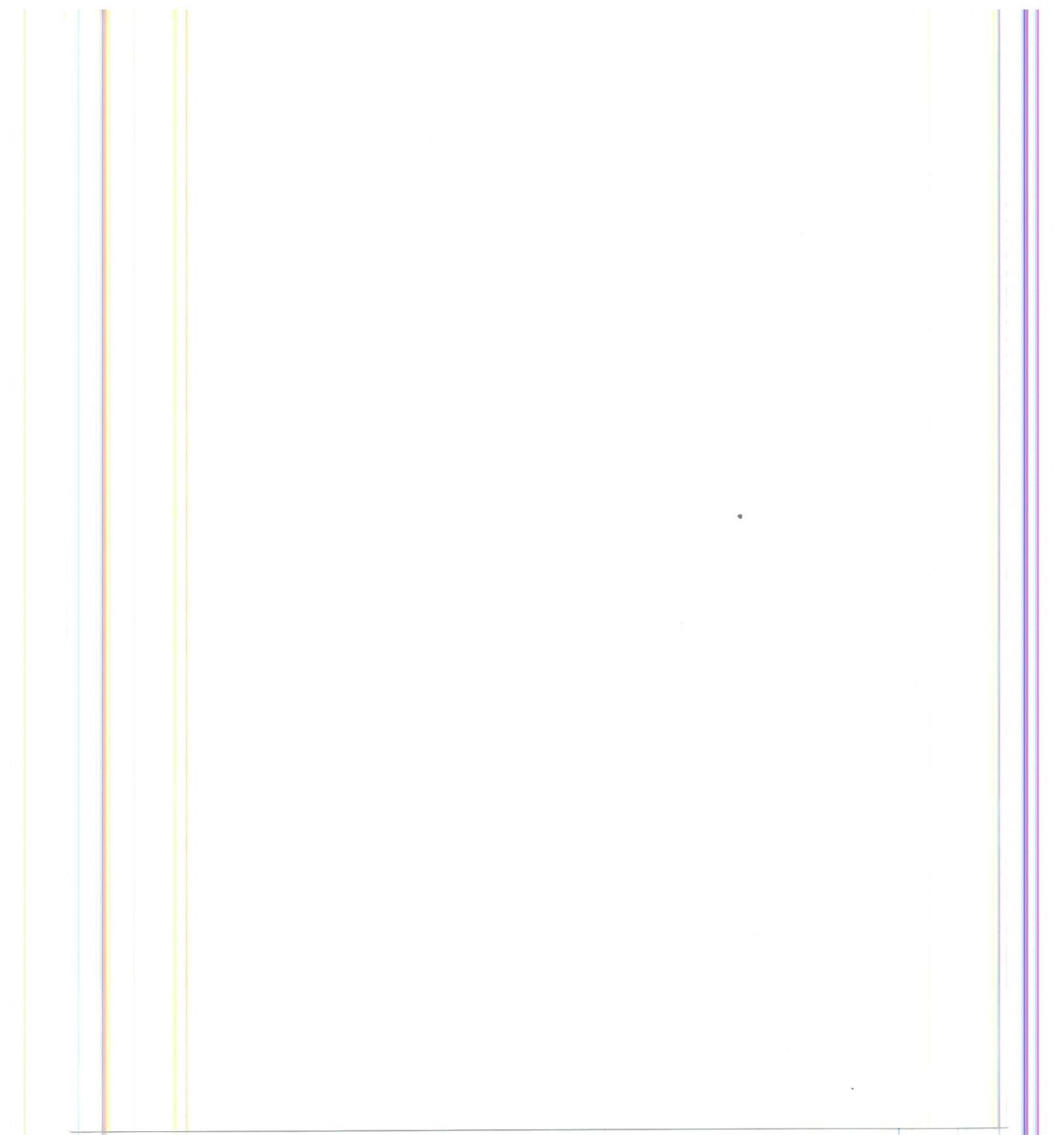
You can control where temporary segments are created for an individual user's statements. By default, temporary segments are created in the SYSTEM tablespace. To change this for a particular user, use the SQL statement:

```
ALTER USER username TEMPORARY TABLESPACE tablespace
```

Temporary segments are allocated independent of a user's storage quota in a tablespace.

Monitoring the Size of Temporary Segments

The size of a temporary segment depends on the storage parameters of the tablespace in which it is created. A tablespace might be created for the sole purpose of allocating temporary segments in which case you can more closely control storage parameters for temporary segments.



17

SECURITY: DATABASE ACCESS

If I carried all the thoughts of the world in my hand, I would take care not to open it.

Bernard Fontenelle

This chapter describes control the DBA has over database access. It describes:

- the differences among the three types of database privileges (CONNECT, RESOURCE, and DBA)
- giving users database privileges using the SQL statement GRANT
- removing database privileges
- changing user attributes, such as passwords, space quotas, and default tablespaces, and using the SQL statement ALTER USER
- using auditing to monitor access to a database.

This chapter explains controlling access to the database as a whole. See Chapter 18 for a discussion of controlling and auditing access to individual database objects and data.

Enrolling Users

A common duty of a DBA is to enable a user of the computer system to use an ORACLE database. In order to use ORACLE data, you must first have access to the computer and its operating system. Though access requirements vary by operating system, typically some sort of operating system identification name and password are required. These are assigned independent from ORACLE and do not provide access to an ORACLE database or programs.

However, some operating system privileges may be required in order to access an ORACLE database; see your *Installation and User's Guide* for a description of any operating system specific requirements for using an ORACLE RDBMS.

For a user to gain access to an ORACLE database, he must have an ORACLE *username* and *password* that are valid for a given database. The data dictionary stores information about every username, such as:

- whether the user has CONNECT, RESOURCE, and DBA privileges
- the user's default tablespace for tables, clusters, and indexes
- the user's default tablespace for temporary segments
- the user's space quota in each tablespace.

At any time a DBA can create new ORACLE usernames using the SQL statement GRANT with the CONNECT option. The DBA may grant one or more database privileges (to multiple users) using the syntax:

```
GRANT { CONNECT | RESOURCE | DBA } TO username1 [, username2 ]  
IDENTIFIED BY password1 [,password2]
```

For example:

```
GRANT CONNECT TO FRIEDA IDENTIFIED BY SANTAFE
```

With one exception, the username and password used for ORACLE (in the example, FRIEDA and SANTAFE, respectively) need have no relationship to the operating system. For example, FRIEDA's operating system ID might be FLAWRENCE and password COAL. However, the DBA can associate an ORACLE username with an operating system ID in order to simplify the logon process (see "Automatic Logins" later in this chapter).

The IDENTIFIED BY clause is required if the CONNECT privilege is being granted. If a username already exists, you can grant additional privileges without the IDENTIFIED BY clause, as in:

```
GRANT RESOURCE TO FRIEDA
```

which requires that FRIEDA is an existing ORACLE username who did not previously have the RESOURCE privilege.

The ORACLE RDBMS accords all ORACLE users the same priority (regarding computer resources). However, database users fall into three classes, corresponding to the database privileges they have been granted; these privileges are described in the next sections.

Users with CONNECT Privilege

A user must have CONNECT privilege in order to access data in an ORACLE database. Every user with CONNECT is identified by both an ORACLE username and a password. ORACLE usernames must be distinct within a database, regardless of their passwords.

The username and password are stored in the data dictionary. The password is stored in encrypted form so that not even a DBA can determine a password by querying the database. Each time a user attempts to connect to a database, the username and password are checked against the dictionary for validity.

It is possible for a user to have the CONNECT privilege only. A user with only CONNECT privilege may:

- access ("connect to") the ORACLE database
- query (look at) other users' data (SELECT from tables and views), if SELECT access has been granted to the user or to PUBLIC
- perform data manipulation operations (INSERT, UPDATE, DELETE) on other users' tables, if the appropriate access has been granted
- create views , synonyms, and database links
- perform table or user exports.

However, the user may not create any tables, clusters, sequences, or indexes.

If you grant CONNECT to a username that previously existed, note that objects belonging to the prior username may still exist and the new user would "inherit" those objects.

If there are multiple databases on one computer and a user requires access to more than one database, then the DBA must grant him CONNECT to each database separately. Access is granted to the database, regardless of the instance used to access the database.

Users with RESOURCE Privilege

If a user has both RESOURCE and CONNECT system privileges, then he has all the privileges associated with the CONNECT privilege and in addition he may create database objects, such as tables, indexes, clusters, and sequences. He may also enable or disable the auditing of his objects and grant to or revoke from other users privileges on his objects.

The RESOURCE privilege may be granted in three ways:

- unrestricted access (to *any* tablespace with *no* quota), as in:

```
GRANT RESOURCE TO STEPHANIE
```

- access to specified tablespaces only, but with no quotas, as in:

```
GRANT RESOURCE ON TABLESPACE RESEARCH TO DEBORAH
```

- access to specified tablespaces with space quotas, as in:

```
GRANT RESOURCE ON TABLESPACE RESEARCH (25000) TO ROBIN
```

The DBA should only grant unrestricted RESOURCE privilege to users whom he does not intend to limit to certain tablespaces or to space quotas. It is usually preferable to grant users access in two SQL statements; first grant CONNECT access, and then grant RESOURCE on particular tablespaces. This allows you to maintain closer control over storage on a username basis.

Users with DBA Privilege

If a user has DBA privilege, he has all the privileges granted by CONNECT and RESOURCE, and in addition he may:

- access any user's data, and perform any SQL statement upon it (except for objects belonging to username SYS)
- GRANT and REVOKE database system privileges to and from users (different from access privileges to database objects)
- create PUBLIC synonyms (which are available to all ORACLE users)
- perform database-wide maintenance operations, such as adding tablespaces or files, bringing tablespaces online and offline, backing up tablespaces, and archiving log files
- control system-wide auditing and table-level auditing defaults
- perform incremental and full database exports.

An ORACLE system may have numerous DBAs, because a DBA is any user with DBA privilege. However, because the DBA has such freedom, the number of users with DBA privilege should usually be limited.

Every ORACLE system is installed with two DBA users: SYS and SYSTEM. (See Chapter 2 for a description of these users.) Because these two users are essential for the operation of an ORACLE database, their access to the ORACLE system should not be altered, other than to change their initial passwords.

DBA duties can be performed by a user logging on to the ORACLE username SYSTEM or by creating additional ORACLE usernames with DBA privilege specifically to perform DBA duties.

DBA usernames may fall into two classes. Within the database, all have DBA privilege, but within SQL*DBA, some may have sufficient privilege to use *system privileged* SQL*DBA commands while some may not. For more information on SQL*DBA system privileged commands and DBA accounts, refer to Chapter 2, Appendix B, and to the *Installation and User's Guide* for your operating system.

Assigning Users Default Tablespaces

SYSTEM is the default tablespace for creating objects and temporary segments. To change the defaults, use the syntax:

```
ALTER USER user
[ IDENTIFIED BY username ]
[ DEFAULT TABLESPACE tablespace]
[ TEMPORARY TABLESPACE tablespace]
```

For example, to change the two default tablespaces for user LEE, who already has been given database access, you might enter:

```
ALTER USER LEE
DEFAULT TABLESPACE TS10
TEMPORARY TABLESPACE TEMP3
```

LEE must also have RESOURCE access to the tablespaces assigned as his defaults for creating database tables or temporary segments.

```
GRANT RESOURCE ON TS10 TO LEE
```

If LEE now creates a table and does not specify a tablespace using the TABLESPACE option, that table will reside in tablespace TS10. All of LEE's temporary segments will also be written in tablespace TEMP3.

If LEE's default tablespace is TS10 but you did not grant LEE RESOURCE on TS10, then upon trying to create a table in TS10 (either by explicitly naming TS10, or by not naming any tablespace and thus requiring the use of the default), LEE will get an error. LEE will not be able to create the table until you grant LEE RESOURCE to TS10.

Setting Tablespace Quotas for Users

The RESOURCE privilege to a tablespace may be granted unconditionally; that is, a user may have unlimited RESOURCE on a tablespace, and thus can use as much space as she desires. This is true if the following syntax is used:

```
GRANT RESOURCE ON TABLESPACE tablespace TO username
```

However, you may impose a limit on the space a user can use in a tablespace, with the syntax

```
GRANT RESOURCE (n) ON tablespace TO username
```

where *n* is the number of bytes to be set as the quota for *username* in tablespace *tablespace*. You should ensure that *n* is a reasonable number for the tablespace (for example, it at least exceeds the default storage parameter INITIAL set for the tablespace).

Thus you could impose a quota of 100,000 bytes on Lee's use of tablespace TS10 with the following SQL statement:

```
GRANT RESOURCE (100000) ON TABLESPACE TS10 TO LEE
```

To see current quotas and the amount of space currently used, query the data dictionary views USER_TS_QUOTAS or DBA_TS_QUOTAS.

Note that temporary segments do not absorb quota, so you need not allow extra quota for temporary segments.

Changing Tablespace Quotas

To change a quota, use the GRANT RESOURCE command with a different number of bytes. If the new quota is lower than the current quota and the user has already exceeded the new quota, no error will occur. Rather, the user cannot allocate more space until his space usage falls below the new quota. From then on he will not be able to exceed the new quota.

Revoking Access to a Tablespace

To revoke access entirely to a tablespace, use:

```
REVOKE RESOURCE ON tablespace FROM username
```

Note that the unrestricted form of the RESOURCE system privilege overrides all other privileges. A user can create objects in any tablespace without limit until it is revoked:

```
REVOKE RESOURCE FROM username
```

Automatic Logins

When enrolling ORACLE users, you may optionally use operating system accounts or IDs as the basis for ORACLE usernames. The benefit is that users enrolled in this way can connect to ORACLE somewhat faster and more conveniently.

The syntax for granting automatic logins follows:

```
GRANT CONNECT TO OPS$sysid IDENTIFIED BY password
```

For example, to enroll a user whose system ID is OPUS, enter:

```
GRANT CONNECT, RESOURCE TO OPS$OPUS IDENTIFIED BY PENGUINS
```

The GRANT syntax is the same except for the prefix "OPS\$" on the ORACLE username, which signifies to ORACLE an automatic login username.

The password (PENGUINS, in this example) can be any arbitrary string, and can be subsequently changed. Passwords for automatic login usernames should be unique; do not, for example, always use XYZ for all automatic login passwords.

Subsequently, when a user who is logged onto the operating system as OPUS connects to ORACLE, he need not explicitly enter either his ORACLE username or password. For example, after he enters the command SQLPLUS, he will see a prompt for username and password. He can press Return, and ORACLE will search the dictionary tables for an automatic login username corresponding to the operating system account name OPUS, find it, and allow OPUS to connect to ORACLE as OPS\$OPUS. Alternately, he can invoke SQL*Plus by entering:

```
SQLPLUS /
```

Users can always connect to ORACLE by specifying a full username and password when desired. For example, a user logged on to the operating system as BILL could issue:

```
CONNECT OPS$OPUS/PENGUINS
```

Thus, even though the current system name is BILL, ORACLE validates the login for OPS\$OPUS as long as the proper ORACLE password is supplied. Therefore, it is important for security that each OPS\$ username have a confidential password. If OPUS changes his operating system password, it has no effect on the ORACLE password.

Because the ORACLE username is the whole name "OPS\$OPUS," all objects created by OPS\$OPUS (tables, views, indexes, etc.) are prefixed by this name. For example, for another user to reference a table, FISH, owned by OPS\$OPUS, he would have to enter:

```
SELECT * FROM OPS$OPUS.FISH
```

Summary of Steps for Enrolling Users

Following is a checklist for enrolling a new user as a valid user of an ORACLE database:

1. Decide whether to link the user's ORACLE username with his operating system account or id.
If you wish to make it somewhat more convenient for the user, give him an OPS\$ username.
2. Assign the user an ORACLE username and password, using an OPS\$ username if desired.

```
GRANT CONNECT TO username IDENTIFIED BY password
```

If the user will not be creating any tables, but will work solely with existing database objects, then you are done (see Step 9).
3. If the user will be creating tables, identify which tablespaces that user will need to access.
4. For each tablespace, decide whether the user can use an unlimited amount of space or should have a tablespace quota.

```
GRANT RESOURCE (n) ON TABLESPACE tablespace TO username
```

(You need not explicitly grant RESOURCE to tablespaces given PUBLIC access, but you can explicitly impose a quota on those tablespaces for a user.)
5. Give the user RESOURCE privilege to each tablespace, indicating a quota in number of bytes, when necessary.

```
GRANT RESOURCE (n) ON TABLESPACE tablespace TO username
```

(You need not explicitly grant RESOURCE to tablespaces given PUBLIC access, but you can explicitly impose a quota on those tablespaces for a user.)
6. If you want the user's default tablespace for creating tables to be other than the SYSTEM tablespace, choose the tablespace.
7. If you want the user's default tablespace for temporary tables to be other than the SYSTEM tablespace, choose the tablespace.
8. Indicate these two default tablespaces using the ALTER USER statement.
9. Tell the user the following information:
 - his ORACLE username and password
 - which tablespaces he has access to, and what his quotas are, if any
 - how to log in to ORACLE
 - how to see the above information in the data dictionary.

Changing Passwords

The ORACLE password for any existing ORACLE user can be changed by that user or the DBA. Either the user or the DBA can use the GRANT CONNECT command supplying the new password, as in:

```
GRANT CONNECT TO username IDENTIFIED BY newpassword;
```

Or the DBA can use the syntax:

```
ALTER USER username IDENTIFIED BY newpassword
```

All usernames are stored in encrypted form in the database; not even DBAs can see actual passwords. The operating system password and ORACLE password are entirely independent.

Dropping Users

To take database privileges away from a user, use the SQL statement REVOKE:

```
REVOKE { CONNECT | RESOURCE | DBA } FROM username
```

for example:

```
REVOKE CONNECT FROM EDISON
```

Only a DBA can change a user's privileges, and he may do so at any time, using the GRANT or REVOKE statements. User's privileges may be increased or decreased by selectively granting or revoking privileges.

If CONNECT privilege is taken away from a user, that user cannot connect to ORACLE unless he is given CONNECT privilege again.

Tables belonging to a dropped user continue to exist, although the user has been dropped from the data dictionary. The DBA can continue to access these tables, as can other users given access to them. The DBA may need to connect as the user in order to drop certain objects, such as synonyms. To do this, temporarily grant CONNECT to the username, drop the objects, and then revoke CONNECT.

If a previously used username is recreated, the restored username "inherits" any currently existing database objects previously created by the username.

Auditing Database Access

Auditing in the ORACLE RDBMS is primarily a security feature for monitoring user activity on a database. It does not journal (duplicate) data that has been updated, deleted, or inserted into the database.

The DBA can enable or disable auditing. If enabled, the DBA can:

- monitor successful and/or unsuccessful attempts to connect/disconnect from the database
- monitor GRANTing and REVOKing of privileges
- enable or disable writing to the audit trail table
- set the default auditing options (if any) for database tables.

For example, the DBA can audit:

- unsuccessful attempts to access the database
- any attempt to access a highly confidential table
- any user's use of the GRANT or REVOKE statements

Enabling System Auditing

By default, auditing is disabled. It can easily be enabled whenever an instance is started. To enable auditing, set the parameter AUDIT_TRAIL in the INIT.ORA parameter file to TRUE (the AUDIT_TRAIL parameter is distributed as FALSE, which turns auditing off). Then start the instance using the edited INIT.ORA file. After the instance has been started, auditing is enabled.

As no type of auditing is automatic, users and the DBA must then use various forms of the SQL statement AUDIT to indicate which actions should be audited.

The DBA may enable system-wide defaults or options, such as setting:

- default auditing options for database objects (such as auditing all successful attempts to ALTER any ORACLE table)
- system-wide database auditing options (such as auditing all unsuccessful attempts to access the database itself).

Users owning tables, views, or synonyms can indicate which actions upon those objects they would like audited (see Chapter 18).

Setting System-wide Auditing Options

A DBA may specify numerous system-wide auditing options to monitor various database activity. Before the system AUDIT statement is used, no system-wide operations are audited.

To specify auditing of system operations, use the following syntax:

```
AUDIT { [ CONNECT ] [ DBA ] [ NOT EXISTS ] [ RESOURCE ] |
        ALL }
        [ WHENEVER [ NOT ] SUCCESSFUL ]
```

where:

CONNECT	Audits connects and disconnects from ORACLE database.
DBA	Audits all uses of GRANT, REVOKE, AUDIT and NOAUDIT statements and CREATE or DROP PUBLIC SYNONYMs or DBLINKs.
NOT EXISTS	Audits all references to objects which result in the "... does not exist" errors. This does not include security violation errors even though such errors appear to the user as error 942 (table or view does not exist). Note that the clause WHENEVER [NOT] SUCCESSFUL is not meaningful.
RESOURCE	Audits all uses of CREATE, ALTER, or DROP TABLE, VIEW, SYNONYM, CLUSTER, SEQUENCE, TABLESPACE, ROLLBACK SEGMENT, or INDEX.

Note that these options are based on the type of operation being performed. They apply to all database users and objects.

For example, the following statement would write records to the audit trail whenever any user successfully connected or executed any DBA operation (that is, executed a SQL statement requiring DBA privilege).

```
AUDIT CONNECT, DBA WHENEVER SUCCESSFUL
```

Listing System-wide Auditing Settings

To see the current system-wide auditing options, query the dictionary view DBA_SYS_AUDIT_OPTS. The result is always a one row table, such as:

```
SELECT * FROM DBA_SYS_AUDIT_OPTS;
C D N R
- - - -
F S - -
```

Seeing Auditing Results in the Audit Trail

All auditing records are written to one database table. Two views are created on this table: DBA_AUDIT_TRAIL and USER_AUDIT_TRAIL (a subset of DBA_AUDIT_TRAIL, this view is described in the next chapter).

The view DBA_AUDIT_TRAIL potentially contains all types of information about past database activity. Depending on the system-wide auditing options selected, this view may record every type of database activity, such as connecting, creating objects, and session information.

The two columns ACTION and ACTION_NAME respectively list the action code and name of the action audited, as in action code 1 which corresponds to CREATE TABLE. Other actions that may be audited include:

```
INSERT  
SELECT  
CREATE CLUSTER  
ALTER CLUSTER  
UPDATE  
DELETE  
DROP CLUSTER
```

Many columns are only filled in by particular actions. For example, the NEW_NAME column is only filled in by a RENAME statement and is otherwise null. Because of the diversity of information potentially stored in the audit trail, users may find it more useful to create views selecting subsets of rows or columns, rather than selecting against the entire audit trail.

Auditing Connections If enabled by the system-wide auditing statement, as in:

```
AUDIT CONNECT WHENEVER NOT SUCCESSFUL
```

then all requested connections (only unsuccessful ones in the example) are recorded in the audit trail. These entries can be seen by querying the view DBA_AUDIT_CONNECT. This view returns a subset of information found in DBA_AUDIT_TRAIL (only information that pertains to user sessions).

Users can see records pertaining to their own sessions by querying a similar view, USER_AUDIT_CONNECT. Note that if a user successfully queries this view, then he will know that auditing of connects is enabled.

Auditing Resource Use If enabled by the system-wide auditing statement, as in:

```
AUDIT RESOURCE WHENEVER NOT SUCCESSFUL
```

then all statements (unsuccessful statements, in this case) requiring RESOURCE privilege are recorded in the audit trail and can be seen by querying the view DBA_AUDIT_RESOURCE. This view returns a

subset of information found in DBA_AUDIT_TRAIL (only information that pertains to any allocation or de-allocation of database storage).

Users can see records pertaining to their own use of space by querying a similar view, USER_AUDIT_RESOURCE.

Auditing DBA Actions

If enabled by the system-wide auditing statement, as in:

```
AUDIT DBA
```

then all statements (unsuccessful and successful statements) requiring DBA privilege are recorded in the audit trail and can be seen by querying the view DBA_AUDIT_DBAA. This view returns a subset of information found in DBA_AUDIT_TRAIL (information that pertains to the attempt to execute DBA SQL statements). Users can query a similar view, USER_AUDIT_DBAA.

Auditing Attempts to Access Non-existent Objects

If enabled by the system-wide auditing statement, as in:

```
AUDIT NOT EXISTS
```

then all statements referencing database objects that do not exist are recorded in the audit trail. These audit entries can be seen by querying the view DBA_AUDIT_EXISTS. This view returns a subset of information found in DBA_AUDIT_TRAIL, information pertaining to SQL statements using invalid names of database objects.

In most cases entries of this type will be generated by honest user error, but this view can be periodically checked if you are concerned about database security or unusual activity and have therefore enabled some system-wide auditing options.

Disabling System-wide Auditing

Use the system NOAUDIT statement to specify which system operations should no longer be audited.

```
NOAUDIT { [ CONNECT ] [ DBA ] [ NOT EXISTS ] [ RESOURCE ] |  
          ALL }  
          [ WHENEVER [ NOT ] SUCCESSFUL ]
```

For example

```
NOAUDIT ALL WHENEVER NOT SUCCESSFUL
```

disables the writing of any records which would have been written on unsuccessful attempts to connect to the database or access or create objects.

The Special "User" PUBLIC

Every ORACLE system automatically enrolls a user called PUBLIC.

This user actually represents a group to which every user with CONNECT access to ORACLE belongs. As members of PUBLIC, ORACLE users may see (SELECT from) all data dictionary tables prefixed with USER and ALL, and any table for which the owner has run the following SQL command:

```
GRANT SELECT ON tablename TO PUBLIC;
```

Any user may grant PUBLIC access to a table or view that is of interest to many users and that he owns or has GRANT access to.

Public Synonyms

Public synonyms are available to all ORACLE users. For example, by entering

```
CREATE PUBLIC SYNONYM synname ON { table | view | synonym }
```

the DBA can create a synonym for a table, view, or another synonym that can be used by all users. Public synonyms are automatically created for the data dictionary views so they are easily accessible.

To see what public synonyms exist, enter:

```
SELECT * FROM ALL_SYNONYMS;
```

Only users with DBA privilege can create public synonyms. An ORACLE user can create an object with the same name as a public synonym. However, upon later referencing that name, the user will see his object and not the object given the public synonym. In order to reference that object, he must use the fully qualified name.

For example, if SCOTT creates a table named DICTIONARY, then the query

```
SELECT * FROM DICTIONARY
```

will return whatever is in the table or view SCOTT called DICTIONARY, rather than the data dictionary view called DICTIONARY. To see the public data dictionary view he would have to query SYS.DICTIONARY:

```
SELECT * FROM SYS.DICTIONARY
```

18

SECURITY: DATABASE OBJECTS

A paranoid is a man who knows a little of what's going on.

William Burroughs

This chapter describes security features affecting database objects. It assumes that users already have access to the database; refer to Chapter 17 for security issues dealing with access to the database itself.

The topics in this chapter include:

- types and levels of access to database objects
- monitoring access to objects through auditing
- SQL statements for controlling auditing of database object.

Different objects can be given different levels of exposure (read-only, update, and delete) to different users. Through auditing, users can record:

- updates or access to their tables
- attempts to alter their database objects.

This chapter will be of interest to DBAs and owners of database objects, as well as application designers.

Access to Table Data

Owners of tables have several options for allowing other users to see or change data in their tables. Access is given with the SQL statement GRANT, and views can be used to limit the data accessed.

Access to All Data in a Table

An owner of a table can give permission to other users to access either all the data in the table or just portions of the data. Permission is given via the SQL statement GRANT. To undo permission, use the SQL statement REVOKE.

Varying types of permission can be given — for example, to read only (SELECT access) or to change data as well (UPDATE and DELETE access). For details, refer to the syntax description of the GRANT statement in Appendix G. SCOTT can use the following statement to give user JENNIFER three types of access to his table EMP:

```
GRANT SELECT, INDEX, UPDATE ON EMP TO JENNIFER
```

After this statement, JENNIFER can refer to table SCOTT.EMP, or for convenience, create her own synonym for SCOTT.EMP.

An option of the GRANT statement (the WITH GRANT OPTION) allows you to pass to other users any privileges you've been granted yourself. For example

```
GRANT SELECT, INDEX, UPDATE ON EMP TO JENNIFER WITH GRANT OPTION
```

allows Jennifer to pass on any of the three privileges (SELECT, INDEX, or UPDATE) on EMP to other users.

If you have been granted the option GRANT INDEX, then you can create or drop any indexes on the table; other users' queries automatically use all applicable indexes. You are implicitly given the authority to drop indexes you have created.

Access to Views (or Selected Data in a Table)

If you want another user to see only a subset of rows or columns of a table, you can create a view on that table and grant the user access to the view, rather than to the table. Thus, for table EMP, you could define two views:

```
CREATE VIEW DEPT10 AS SELECT *  
FROM EMP WHERE DEPTNO = 10
```

This view selects a subset of *rows* from table EMP, whereas the following view selects a subset of *columns*:

```
CREATE VIEW SALES AS SELECT ENAME, EMPNO, MGR  
FROM EMP WHERE JOB = 'SALESMAN'
```

Then you can grant access to the views to the appropriate persons:

```
GRANT SELECT, UPDATE, INSERT ON DEPT10 TO DEPT10AA  
GRANT SELECT, UPDATE ON SALES TO SALESMGR
```

Access privileges are different for views than they are for tables. For example, it does not make sense to INDEX a view. Privileges that can be granted on views are SELECT, INSERT, UPDATE, and DELETE.

In addition, there are some restrictions on DML statements that can be performed on views:

- You cannot UPDATE, INSERT, or DELETE from views whose columns are based on expressions, joins, or set operations (UNION, INTERSECT, MINUS)
- Owners of a view on a table (say User-A owns VIEWA on User-B's TABLEB) cannot GRANT SELECT on the view unless they have been given GRANT on the underlying table (in this case, TABLEB).

You can indicate WITH GRANT OPTION for views, just as for tables. If a view includes a table that does not belong to you, you must have GRANT OPTION for the privilege on that table in order to grant that privilege on the view.

Access to UPDATE Columns

If you want another user to be able to update only a subset of columns of a table, you have two choices. You can use the GRANT statement to name the columns along with the name of the table:

```
GRANT SELECT, INDEX, UPDATE ON EMP (ENAME, MGR, HIREDATE)  
TO JENNIFER WITH GRANT OPTION
```

Or, if the user will not need to DELETE, then you can create a view on that table based on only those columns and grant the user access to the view.

Access to Sequence Numbers

The creator of a sequence can give other users the right to use the sequence using the GRANT statement with the SELECT and ALTER options. Like other forms of the GRANT statement, he may also give the GRANT OPTION, so those users can pass on grants to other users.

Revoking Access Privileges

To take table privileges away from users, use the REVOKE statement to explicitly revoke a privilege (or use the keyword ALL):

```
REVOKE INDEX ON EMP FROM JENNIFER
```

Now JENNIFER can no longer create indexes on EMP, although any indexes she had created are not affected. If JENNIFER had passed on the option to create indexes to other users, their privileges are also revoked, unless the privileges were also granted by another user.

You revoke access to a view just as for a table.

To revoke selective access (access granted to certain columns for a table), you must revoke access on the whole table (and then re-grant access on certain columns if you like):

```
REVOKE UPDATE ON EMP FROM MARGY  
GRANT UPDATE ON EMP (ENAME) TO MARGY
```

Auditing Database Objects

If a DBA has enabled system-wide auditing, then ORACLE users owning tables or views can use the SQL AUDIT statement to request the following auditing activities:

- audit successful and/or unsuccessful attempts to access tables or views
- selectively audit different types of SQL operations
- control the level of detail recorded in the audit trail (session vs. access).

Similar to the SQL GRANT and REVOKE statements, the auditing options you may enable on database objects (tables and views) are based on DML activity, such as SELECT, UPDATE, INSERT, or DELETE. For example, the following statement would audit successful alterations to the EMP table (such as adding a new column):

```
AUDIT ALTER ON EMP BY ACCESS WHENEVER SUCCESSFUL
```

After auditing options are chosen, the resulting auditing entries are written to a data dictionary table, SYS.AUDIT_TRAIL, also called the "audit trail." Users can see auditing entries directly by querying the view USER_AUDIT_TRAIL, which is created on this table.

Who Can Use Auditing?

The DBA enables or disables auditing and can set system-wide defaults for auditing. If the following circumstances are true, users can use auditing:

- auditing is enabled
- they have RESOURCE privilege on the database
- they own tables and views
- they have AUDIT privilege on other user's tables or views.

These users can set auditing options for database objects they own or have access to.

Setting Table Auditing Options

```
AUDIT { table_option [ , table_option ] ... ALL }
ON { table | DEFAULT }
[ BY { SESSION | ACCESS } ]
[ WHENEVER [ NOT ] SUCCESSFUL ]
```

The valid *table_options* are:

```
ALTER
AUDIT
COMMENT
DELETE
GRANT
INDEX
INSERT
LOCK
RENAME
SELECT
UPDATE
```

If you do not specify any WHENEVER clause, then regardless of whether an operation is successful or unsuccessful, an entry is made; the effect is as if you used an option called BOTH (successful and unsuccessful).

See the following section for a discussion of BY SESSION and BY ACCESS.

For example:

```
AUDIT ALTER, SELECT, UPDATE ON SCOTT.EMP
WHENEVER SUCCESSFUL
```

Auditing by Session or by Access

When choosing auditing options for a database table, you must choose to audit at one of two levels: by session or by access. Your choice determines the number of audit entries that will be written during a given user's ORACLE session.

by session

Exactly one record is written per user per session. (An ORACLE session begins at CONNECT and ends at DISCONNECT.) You choose which SQL statements you want audited; the rows in the audit trail table record how many times such statements executed successfully or unsuccessfully. Thus, each row is a summary record for one user's session.

by access

One record per access to a database object is written per user. Thus, many, one, or no records can result from one user's ORACLE session.

Note that for a DML operation such as in INSERT or DELETE, the audit trail entry is written (by access) or updated (by session) at completion of the operation's parse phase and before execution. Thus, if the statement is rolled back, that fact is not reflected in the audit trail.

Auditing Views

When a view is created, its auditing options are set to the union of the DEFAULT table auditing options and the auditing options of all base tables referenced by the view. The "union" of BY ACCESS and BY SESSION is BY ACCESS.

Default Table Auditing

When a table is created, its auditing options default to those specified on a pseudo table referred to as DEFAULT. Only a DBA can specify these options, using the keyword DEFAULT rather than a tablename in the AUDIT statement, as in:

```
AUDIT ALTER, RENAME, GRANT ON DEFAULT
```

Be careful when specifying default options that you request only the information you want; by specifying too much auditing you may be overwhelmed with information of little importance, and that information may require a fair amount of database storage.

To see the default settings, query the one-row data dictionary view ALL_DEF_AUDIT_OPTS. This view is very similar to USER_TAB_AUDIT_OPTS, described in the next section.

If the DBA does not specify default table auditing options, then no table auditing occurs unless it is explicitly requested by a table owner. Auditing options specified by a table owner override default auditing options.

Listing Current Auditing Options

To see what auditing options are in effect for your own tables, query the dictionary view `USER_TAB_AUDIT_OPTS`.

```
SELECT * FROM USER_TAB_AUDIT_OPTS

TABLE_NAME    O_TYPE    ALT AUD COM DEL GRA IND INS LOC REN SEL UPD
-----        -----    -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
EMP          TABLE     -/- -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
ACCOUNTS     TABLE     S/S -/- S/S S/S S/S -/- S/S A/- -/- S/S S/S
PAYABLES      TABLE     A/S -/- -/- /S -/- -/- A/S A/- -/- A/S A/S
```

You may see similar information in the views `ALL_TAB_AUDIT_OPTS` and `DBA_TAB_AUDIT_OPTS`; these views have an additional column showing the owner of the object.

The last eleven columns correspond to the list of actions which may be audited (see "Setting Table Auditing Options" earlier). Their values show the current auditing setting, if any. Each column contains two characters per table_option. The first character describes what to do when the audited action succeeds and the second character describes what to do when the audited action fails. The character codes are:

- Do nothing (equivalent to no auditing).
- A Write one audit record per access.
- S Write one audit record per session.

To change these values, do not update this table directly; instead, use the SQL statement `AUDIT` to set the preferred auditing choices.

Seeing Auditing Results in the Audit Trail

No matter which variation of the `AUDIT` statement is used to enable auditing, all auditing entries are written to the same database table. A user who has enabled auditing on a database table can see all resulting auditing entries by querying the view `USER_AUDIT_TRAIL`. These entries record SQL DML statements that referenced that object.

In the view, column `SES_ACTIONS` is an 11-character array whose entries correspond to the 11 table_options. The values that appear are a summary by user-session of any table activity. Actions are encoded as follows:

- No session-auditable activity occurred.
- S The action successfully executed.
- F The action unsuccessfully executed.
- B The action both successfully and unsuccessfully executed (in multiple executions).

The audit trail table may grow rapidly, depending on the actual auditing options in effect. Data in this table may be deleted by the DBA; it is the sole data dictionary table whose data may be directly altered by a DBA.

Disabling Table Auditing

Use the table NOAUDIT statement to specify operations on a particular table that should no longer be audited:

```
NOAUDIT { table_option [, table_option ]... | ALL }
ON { table | DEFAULT }
[ WHENEVER [ NOT ] SUCCESSFUL ]
```

For example:

```
NOAUDIT ALL ON emp
```

The NOAUDIT statement syntax is similar to that of the AUDIT statement. The main difference is that it has no BY clause; the specified auditing activity is terminated whether it is BY ACCESS or BY SESSION. Options specified in a NOAUDIT statement need not be audited currently.

The Function USERENV

The USERENV function returns information about a particular ORACLE session. The ORACLE RDBMS uses this function to determine values to insert into AUDIT_TRAIL. The function USERENV is useful for constructing views that constrain access to users logged on using certain physical terminals. For example:

```
CREATE VIEW MYVIEW AS
SELECT * FROM MYSALARY
WHERE USERENV('TERMINAL') = 'RTA0';
```

The actual specification of the terminal ID will vary according to the format for your operating system. Once you create the view, you must also grant access to it to selected ORACLE users:

```
GRANT SELECT ON MYVIEW TO EVELYN;
```

After both these statements, EVELYN will only see the rows in MYVIEW when she is logged on to terminal RTA0. Though she can access MYVIEW from other terminals, it will not return any rows.

19

IMPROVING PERFORMANCE OF APPLICATIONS

*The more human beings proceed by plan
the more effectively they may hit by accident.*

Friedrich Durrenmatt

This chapter's focus is how ORACLE users can use ORACLE RDBMS features most optimally. When you understand in detail how ORACLE works, you can begin to "fine tune" your applications for best performance.

For example, it describes the performance implications of the wording of SQL statements. You can often improve performance by rewording SQL queries to make use of more selective indexes.

Other topics covered in this chapter include:

- using indexes and clusters effectively
- optimizing various SQL statements
- using array processing.

Individual users have many ways to improve their own ORACLE applications, independent of other users. Among the most valuable tools are indexes and clusters.

While this chapter describes actions a single user or designer can take, Chapter 20 describes actions which can be taken on a broader basis to improve ORACLE performance, specifically for multiple and concurrent users. These steps include adjusting INIT.ORA parameters and setting database defaults.

In addition, operating system considerations may help ORACLE performance; details on this type of tuning are found in the *Installation and User's Guides*.

Note: The information contained in this chapter pertains to the current version of the ORACLE RDBMS at the time of writing; this information is subject to change in future releases.

The Importance of Good Relational Table Design

The first step of performance tuning is to *know your data*. Good performance instructions cannot counteract the effects of poor database design. The performance gains an application can achieve depend largely upon how well the base database tables are defined and stored.

Given the same set of data to be stored, the sets of tables and storage parameters that may define it (data schema) are virtually unlimited. The set of tables actually used can serve to highlight the relational advantage (if designed carefully) or can lead to bad performance and design traps (if designed poorly).

A good understanding of relational theory and normalization, and their practical application (both of which are beyond the scope of this manual) will go far toward helping designers and users achieve the flexibility and high performance of a well-designed relational database application.

As in many complex environments, there are often many tradeoffs to consider or on which to experiment. Sometimes you may deviate from accepted relational theory or wisdom to achieve a particular goal. One of the advantages of relational databases is that many changes can be effected quickly, with little to no effect on applications.

Aside from proper table design, users can notice substantive performance gains by using the indexing and clustering features.

Operator Precedence

The use of parentheses to clarify the meaning of more complex SQL statements is recommended. For example, without parentheses the meaning of the following SQL statement is not obvious

```
SELECT ENAME, DEPTNO  
FROM EMP  
WHERE JOB = 'CLERK'  
AND DEPTNO = 10  
OR DEPTNO = 20
```

because a user could interpret it as either of the two requests:

1. "Display names and departments of all clerks in department 10 and anyone in department 20" or
2. "Display names and departments of all clerks but only if they are in department 10 or department 20."

Assume we want to see number 2: the names and department numbers of all clerks who work in either department 20 or department 30. If we add punctuation we can change the meaning of the query (and thus its output) as well as clarify our intention. If we use no punctuation

```
SELECT ENAME, DEPTNO  
FROM EMP  
WHERE JOB = 'CLERK'  
AND DEPTNO = 20  
OR DEPTNO = 30
```

the result is all clerks who work in Dept 20, and all employees in Dept 30:

ENAME	DEPTNO
ALLEN	30
WARD	30
MARTIN	30
TURNER	30
ADAMS	20
JAMES	30

If we add parentheses around (DEPTNO = 20 or DEPTNO = 30)

```
SELECT ENAME, DEPTNO  
FROM EMP  
WHERE JOB = 'CLERK'  
AND (DEPTNO = 20  
OR DEPTNO = 30)
```

the result is all clerks who work in Dept 20 or Dept 30 (the desired result):

ENAME	DEPTNO
ADAMS	20
JAMES	30

If we add parentheses in a different place:

```
(JOB = 'CLERK' OR DEPTNO = 20) :
```

```
SELECT ENAME, DEPTNO
FROM EMP
WHERE (JOB = 'CLERK'
AND DEPTNO = 20)
OR DEPTNO = 30
```

the result is the same as the result with no punctuation, but it is not as ambiguous — all clerks who work in Dept 20 and all employees in Dept 30:

ENAME	DEPTNO
ALLEN	30
WARD	30
MARTIN	30
TURNER	30
ADAMS	20
JAMES	30

Using the SQL Language Effectively

The SQL data language is a rich, flexible, and powerful language. If you are imbedding SQL statements in application programs, examine those programs to see whether you are using the programming language to accomplish functions that SQL might better provide. It is usually advantageous to perform operations in SQL whenever possible, rather than in the host language.

Also, become familiar with the numerous SQL language functions, such as DECODE, SUBSTR, and INSTR. You may be surprised at the ease with which you can accomplish fairly complex queries or data manipulations.

For more information on the SQL language, refer to the *SQL Language Reference Manual*.

Optimizations Automatically Performed

This section briefly describes how some ORACLE optimizations work, over which you have little or no control. Sections following this one describe various situations where you may be able to have an impact by re-wording certain SQL statements.

Using DISTINCT

Given a SQL statement including a query with one or more DISTINCT items requested, ORACLE sorts the rows, discarding duplicates as they are encountered. Indexes are not used to process the DISTINCT clause (though they may still be involved in other parts of the query).

Using GROUP BY

GROUP BY queries are processed similarly to DISTINCT queries; given a GROUP BY query, ORACLE sorts the rows, aggregating groups during the sort, and discards duplicates as they are encountered. Indexes are not used for GROUP BYs (though they may still be involved in other parts of the query).

Subqueries

Certain subqueries are automatically transformed into joins. Queries using IN can sometimes be transformed into joins. For example

```
SELECT * FROM EMP  
WHERE EMP.DEPTNO IN  
(SELECT DEPTNO FROM EMP)
```

can be processed as the following logical query, which is *not* in valid SQL syntax:

```
SELECT EMP.A FROM EMP, D:  
(SELECT DISTINCT DEPTNO FROM DEPT)  
WHERE D.DEPTNO = EMP.DEPTNO
```

Other similar transformations are performed automatically.

Single Row Query Paths

When a table in the FROM clause is guaranteed to point to a single row, that table is processed first in a join. Examples of such paths include:

- unique index paths with WHERE = constant
- single set views (such as SELECT AVG(SAL) FROM EMP)
- ROWID paths.

Writing SQL Statements to Take Advantage of Indexes

ORACLE RDBMS accesses table data in one of two ways:

- using a full table scan
- using index(es).

Using indexes is almost always preferable to using a full table scan. DML statements (in particular SELECT, UPDATE, and DELETE) often contain clauses identifying which records should be included or excluded; for example, update the salary for all employees in development, or select birthdays of anyone born in November. An index allows more immediate access to just the desired rows.

A full table scan is preferable when you know that more than about 25 percent of the records in the queried tables will be selected; in this case using the index to find all of the records only adds the overhead of reading the index in addition to the table.

An index can be used if:

- it is referenced in a predicate. A *predicate* is each portion of selection criteria used to include or exclude rows from a result. For example, the following WHERE clause contains two predicates:

```
WHERE DNAME = 'DEVELOPMENT'  
AND SEX != 'FEMALE'
```

- the indexed column is not modified by a function or arithmetic operation.

An index will be used if the optimizer decides it is appropriate (several following sections describe the choices ORACLE makes based on the set of circumstances, many of which a user can control).

An index will not be used if:

- there is no WHERE clause
- the predicate modifies the indexed column in any way (via a function or arithmetic expression)
- the search is explicitly for records with NULL or NOT NULL values in the indexed column (that is, the predicate contains either IS NULL or IS NOT NULL).

If the column must be modified to meet the selection criteria, as in the following predicates, an index cannot be used:

```
WHERE SAL * 12 = 24000  
WHERE SAL + 0 = 500  
WHERE ''||ENAME = ' Smith'  
WHERE SUBSTR(ENAME, 1,1) = 'S'
```

One exception exists to the "indexed column cannot be modified" rule: the special optimization to find the largest or smallest value in the table allows the SELECT clause to contain an expression with

```
{ MIN | MAX } ( col { + | - } constant )
```

and no other columns, as in:

```
SELECT 2 * MAX (SAL+1) FROM EMP
```

Creating Effective Indexes

Concatenated indexes can be a powerful performance tool, especially when they allow certain queries to be satisfied by reading only index blocks and not data blocks.

For the following examples, assume the following concatenated index has been created:

```
CREATE INDEX INDNAME_CITY_STATE ON  
EMPLOYEE (LAST_NAME, CITY, STATE)
```

Because the WHERE clause names the same three columns that were named in the index (regardless of the order the columns were named), the index will be used in the query:

```
SELECT ...  
FROM EMPLOYEE  
WHERE LAST_NAME = 'SMITH'  
AND CITY = 'FREDERICK'  
AND STATE = 'MD'
```

If only the first column of the index is used in the WHERE clause, the index will still be used but the index will not be as selective as in the previous example.

```
SELECT ...  
FROM EMPLOYEE  
WHERE LAST_NAME = 'JONES'
```

The entire query can be satisfied by the index if all selected columns are indexed, as in:

```
SELECT CITY, STATE  
FROM EMP  
WHERE LAST_NAME = 'SMITH'
```

In the next two examples, the index cannot be used, because the WHERE clause does not name the first column in the index:

```
SELECT ...  
FROM EMPLOYEE  
WHERE CITY = 'BLACKSBURG'  
AND STATE = 'MD'  
  
SELECT ...  
FROM EMPLOYEE  
WHERE STATE = 'DC'
```

A given query can be worded in different ways to use or avoid the use of an index. For the following examples, assume that the column HIREDATE is defined as DATE and has a nonunique index.

The query

```
SELECT * FROM EMP  
WHERE TO_CHAR(HIREDATE, 'Month dd, yyyy') = 'January 14, 1986'
```

modifies the HIREDATE column with the TO_CHAR function; the index on HIREDATE is not used. Whereas, in the query

```
SELECT * FROM EMP  
WHERE HIREDATE = TO_DATE('January 14, 1986', 'Month dd, yyyy')
```

the constant (January 14, 1986) is converted to a date, and so ORACLE will use the index to do the search. In general, it is better practice to make constants used as selection criteria match the datatypes defined for the table.

This query would also use the index:

```
SELECT * FROM EMP  
WHERE HIREDATE = '14-JAN-86'
```

Single Indexes

In a simple query on one table, where a predicate refers to an indexed column, once ORACLE has identified the index and has decided to use it, ORACLE searches the index for the value(s) of interest, rather than scanning the table. (A *scan* is a sequential lookup through a table, whereas a *search* is a search using an index.) Thus, both a full table scan and an index scan are avoided.

For the following query

```
SELECT EMPNO, ENAME FROM EMP  
WHERE JOB = 'CLERK'
```

once ORACLE finds that the JOB column has an index (either unique, which is unlikely in this case, or nonunique), it will search for the desired value in the index ('CLERK'), and return all records with the value 'CLERK'.

Indexes and Null Values

When possible, define columns of tables as NOT NULL; this will allow index use on a slightly greater number of queries. Indexes are not used if the predicate contains either of the phrases:

```
IS NULL  
IS NOT NULL
```

However, if you are interested in obtaining the not null values (but not the null values), you can usually write the SQL statement so that it will use an index. This can be done to return rows where a column has a value, not the columns where the value is null.

For example, if you wanted to see all employees who make a commission (whose commission is NOT NULL), you could use either of the queries:

Query 1:
SELECT * FROM EMP
WHERE COMM IS NOT NULL

Query 2:
SELECT * FROM EMP
WHERE COMM >= 0

Assuming an index exists on the COMM column, it is not used for Query 1, but it would be used by Query 2 to get the results without performing a full table scan. However, if most records in the EMP table (more than 25%) have a value for COMM, then Query 1 is preferred. Query 2 is only preferred if most records have a null value for COMM.

Multiple Indexes on One Table

A query with two or more predicates on the same table can use multiple indexes if:

- the indexes are nonunique column indexes
- the predicates are equalities.

Data returned from each index's search is "merged" with previous results to obtain the final result. Multiple indexes are not used for "bounded range and equality" predicates, such as (DEPTNO != 10) in:

```
SELECT *
FROM EMP
WHERE JOB = 'MANAGER'
AND DEPTNO != 10
```

Rather, the index for the equality (JOB = 'MANAGER') will drive the query.

Choosing Among Multiple Indexes

In queries with multiple indexes available but no clear preference, ORACLE chooses the driving index based on the types of index (unique or nonunique) and the column characteristics.

When both a unique and nonunique index are available, ORACLE uses the unique index and ignores the nonunique index, thus avoiding the "merge." For example, assuming a unique index on EMPNO and a nonunique index on SAL, in the following query:

```
SELECT ENAME
FROM EMP
WHERE SAL = 3000
AND EMPNO = 7902
```

ORACLE will use only the index on EMPNO. If a row is found with EMPNO 7902, the actual SAL field is examined to see if it is 3000, rather than using the index on SAL.

A maximum of five indexes are merged. If more than five criteria (WHERE predicates) exist in the SQL statement, as many as five indexes are merged, but after five, all remaining data is "manually" checked to extract records meeting the remaining criteria.

Suppressing the Use of Indexes

Since all indexes that are appropriate are merged, it is occasionally possible that the use of an index can decrease performance (because the index would use a query path that was not the most optimal for the particular query). Or, since up to five indexes are used in a query, if you are writing a SQL statement with predicates referencing more than five indexed columns, then you may wish to suppress the use of the "least valuable" (least selective) indexes.

To suppress index use, simply use a "dummy" function or expression with the column whose index you do not want to invoke. Typically you would either add a zero to a numeric column or concatenate a null string to a character column, as in:

```
SELECT ENAME, DEPTNO, SAL  
FROM EMP  
WHERE DEPTNO + 0 = 20  
AND    ENAME = 'SMITH'
```

Optimizing Various SQL Statements

Various optimizations are automatically performed on your behalf by the ORACLE RDBMS; furthermore, by carefully wording certain SQL statements, you can improve their performance. To compare the paths taken by various statements, you can use the EXPLAIN statement; for documentation on EXPLAIN, refer to the online documentation contained in the file EXPLAIN.DOC, found on the distribution media.

Optimizing Queries (SELECTS)

On a small table, ORACLE will perform reasonably fast sequential scans; indexes are unnecessary. However, when retrieving less than 25% of the rows in a table with many rows, then the use of indexes or clusters can considerably reduce the query time. Specifically, if columns in large tables named in the WHERE clause are indexed or clustered, the search will be much faster than if they are not.

When many columns are named in the WHERE clause (indexed, clustered, or neither), ORACLE chooses which WHERE conditions to test first (that is, which column criteria will "drive" the query).

For example, evaluating one WHERE clause first may immediately eliminate 75% of the rows from a large table, while evaluating the other clause first might eliminate only 15% from a different large table.

ORACLE uses various internal algorithms to choose a starting point (called the "driving" column or condition) from which to proceed with all the testing against WHERE clauses.

Note that the optimizer algorithms may change over time and between ORACLE versions. As Oracle Corporation continues its research and development of optimization, it is possible that different optimization paths might be chosen for the same query in different versions of ORACLE.

General rules of thumb comparing various paths include:

<i>These paths are faster</i>	<i>than these paths</i>
indexed columns	unindexed columns
unique indexes	nonunique indexes
ROWID (= constant)	any other search
bounded range	unbounded range
pattern match like 'x%'	pattern match like '%x%

Optimizing NOTS

ORACLE does not use indexes for predicates containing not equal clauses (as in != or NOT =), though it may use an index on another predicate. For example, in

```
WHERE X != 7 AND Y = 8
```

though no index is used for column X, an index could be used for column Y. Usually in queries with "NOT =" the number of rows returned is greater than the number of rows skipped. For ORACLE to read the index and then the table requires extra I/O. Thus it is usually faster to do a full table scan than use an index.

When the predicate contains NOT in conjunction with other operators, however, ORACLE transforms the predicate into one with which indexes may be used, as follows:

<i>This:</i>	<i>transforms to this:</i>
NOT >	<=
NOT >=	<
NOT <	>=
NOT <=	>

Optimizing ORS

SQL statements using OR clauses use indexes under some circumstances. Generally, optimization will occur if the columns in the OR predicates are indexed, but optimization may not occur if some columns are indexed but some are not.

Given a query such as

```
SELECT *
FROM EMP
WHERE DEPTNO = 10 or JOB = 'CLERK'
```

and indexes exist on both the columns DEPTNO and JOB, ORACLE will use both indexes and perform something similar to the union of the two queries:

Query 1:	Query 2 :
SELECT *	SELECT *
FROM EMP	FROM EMP
WHERE DEPTNO = 10	WHERE JOB = 'CLERK'
	AND DEPTNO != 10

Knowing this, it is best to put the most specific index first in the predicate list, and the predicate that passes the most records last. This will minimize the number of checks for "not equal to."

OR predicates are not used for optimization in the following cases:

- when the SQL statement contains a CONNECT BY
- when the SQL statement contains an outer-join
- when ORACLE determines that using indexes will not optimize the query.

The optimization also applies to IN clauses which translate to ORs. Thus, the predicate

```
DEPTNO IN (X, Y, Z)
OR DEPTNO = Z
```

means the same as:

```
DEPTNO = X OR DEPTNO = Y OR DEPTNO=Z
```

The ORACLE Sort/Merge Routine

The sort/merge routine of ORACLE RDBMS speeds performance on several operations, such as CREATE INDEX, SELECT DISTINCT, ORDER BY, GROUP BY, and certain joins. Simply described, incoming data which must be ordered is first internally sorted into several sets of ordered runs, and then these runs are merged into a final, sorted result.

Three INIT.ORA parameters influence the sort/merge routine; see Appendix D for descriptions of the parameters beginning with SORT.

Two work areas influence the sort/merge routine:

internal	This size is set by the INIT.ORA parameter SORT_AREA_SIZE.
----------	---------------------------------------------------------------

external	<p>This area is used for storage of runs before and during the merge. External work area is allocated in a temporary segment. In the worst case, an external work area of twice the size of the column(s) being sorted may be required to accomplish the final merge pass.</p>
Optimizing ORDER BY	<p>Though there may be several ways to word a SQL statement to achieve the ordering you desire, the only way to guarantee ordering is to use the ORDER BY clause at the end of a SQL statement.</p> <p>Results which appear ordered, but were not obtained using an ORDER BY clause, may not be returned in the same order in future releases. Thus, you should explicitly request ordering via the ORDER BY clause.</p> <p>For a query with an ORDER BY clause, first the data that meets the selection criteria is extracted, and then it is sorted using either available indexes or the sort/merge routine.</p>
Optimizing GROUP BY	<p>When grouping data, you should eliminate rows not meeting selection criteria using a WHERE clause, so extra rows do not incur processing. For example, the query</p> <pre>SELECT JOB, AVG(SAL) FROM EMP WHERE JOB != 'PRESIDENT' AND JOB != 'MANAGER' GROUP BY JOB</pre> <p>is better than the query</p> <pre>SELECT JOB, AVG(SAL) FROM EMP GROUP BY JOB HAVING JOB != 'PRESIDENT' AND JOB != 'MANAGER'</pre> <p>for two important reasons:</p> <ol style="list-style-type: none"> 1. Because Query 2 has no WHERE clause, all rows are grouped and incur HAVING processing. This includes rows which might have been eliminated with a WHERE clause and which could have been "discarded" earlier. 2. HAVING clauses are intended to eliminate grouped data based on group criteria (such as less than a minimum or greater than average). Though their syntax is similar to that of WHERE clauses, HAVING clauses cannot always (and therefore should not) be used interchangeably with WHERE clauses.

Optimizing Joins

The following are some general rules for optimizing join performance:

- Columns used in join clauses should be indexed.
- SQL statements should be worded so as to use indexes, rather than suppress their use (that is, watch the wording of the predicates so that the use of indexes is not inadvertently suppressed).
- Joins should be driven from tables returning fewer rows rather than tables returning more rows.

Unindexed Joins

If no indexes exist on the joined columns in a join, a "sort-merge" join is performed, as long as the following are true:

- There is no other query path usable by ORACLE. (If ORACLE determines that indexes can be used to execute the query, it will use indexes and not the sort/merge.)
- The join clause must be of the form

```
tab_a.expression = tab_b.expression
```

as in:

```
tab_a.col_a + 1 = to_number(tab_b.col_b)
```

For example, the join may not be a greater-than-or-equal. Outer-equijoins, however, may be performed by sort/merge.

In a sort-merge join, each table to be joined is processed separately; records that pass a table are sorted. Then the sorted lists are merged, based on the expressions forming the join.

If there are additional selection predicates (such as WHERE DEPTNO = 20) put the most selective predicates last in the predicate list.

Indexed Joins

If only one of the tables being joined has a usable index, the other table is usually the driving table. For example, if EMP.DEPTNO is indexed and DEPT.DEPTNO is not indexed, then in both the following queries, the DEPT table is the driving table:

```
SELECT ENAME, DNAME  
FROM DEPT, EMP  
WHERE EMP.DEPTNO =  
      DEPT.DEPTNO  
AND JOB != 'SALESMAN'
```

```
SELECT ENAME, DNAME  
FROM EMP, DEPT  
WHERE EMP.DEPTNO =  
      DEPT.DEPTNO  
AND JOB != 'SALESMAN'
```

Exceptions include when a query path is guaranteed to be a single row query path.

If indexes exist on both the joined columns or on subsequent selection predicates, then ORACLE ranks the access paths based on their selectivity, in order to choose the optimal path. In general, ORACLE considers:

- which predicates can use indexes
- which indexes are unique.

In constructing the query, the following may also affect performance:

- the number of records in the tables queried
- the distribution of data indexed by a nonunique index.

The actual query path ORACLE chooses is a result of weighing all paths against each other with a ranking scheme, shown in Figure 19-1. The rankings determine which table will be the driving table.

Note that:

- Every predicate joined by AND is evaluated separately and is scored in the sequence shown in Figure 19-1, from lowest (fastest) to highest (slowest).
- *Concatenated index* means either an index composed of multiple columns in a table or a cluster key.
- Joins are scored on the number of tables that will be joined without an index and the number of Cartesian products.
- If ranks are equal, then, as when there are no indexes, ORACLE uses the last table in the FROM clause as the driving table.

Thus, you should list large tables with the smallest number of qualified rows last in the FROM clause, and when writing a multi-table join (3 or more tables), list the join clause for the pair of tables resulting in the smallest result set last in the list of join clauses.

If a certain table has a low score for a search, but can only be joined to other tables without the use of indexes, then another table that can join with indexes will be chosen as the driver. The join score always dominates the cumulative score.

FIGURE 19-1
Query Paths Ranked
in Order of Speed

The *lower* the rank, the *faster* the path.

Rank	Path
1	ROWID = constant
2	unique indexed column = constant
3	entire unique concatenated index = constant
4	entire cluster key = corresponding cluster key in another table in same cluster
5	entire cluster key = constant
6	entire nonunique concatenated index = constant
7	nonunique index = constant
8	entire concatenated index = lower bound
9	most leading concatenated index specified
10	unique indexed column BETWEEN low value AND high value, or unique indexed column LIKE 'C%' (bounded range)
11	nonunique indexed column BETWEEN low value and high value, or nonunique indexed column LIKE 'C%' (bounded range)
12	unique indexed column or constant (unbounded range)
13	nonunique indexed column or constant (unbounded range)
14	sort/merge (joins only)
15	MAX or MIN of single indexed column
16	ORDER BY entire index
17	full table scans
18	unindexed column = constant, or column IS NULL, or column LIKE '%C%' (full table scan)

Array Processing

The array processing feature allows multiple rows to be processed at once. Array processing can yield as much as ten times the performance of single row operations. Array sizes of 100 or more rows at a time are most optimal. Several ORACLE utilities use arrays to decrease execution time.

Users can see performance gains in INSERTS and SELECTS when using arrays in user-written programs with embedded SQL. For a description of datatypes and call syntax, see the guide pertaining to the programming language of interest, as in the *Pro*FORTRAN User's Guide*.

Avoid Reparsing and Rebinding

Before a SQL statement can be executed, a cursor must be opened, the statement parsed by the kernel, and any host variables bound to the cursor. The time involved in these preparations is less than a second or two, and is usually negligible.

Query preparation can consume a significant percentage of elapsed time if a transaction requires the execution of several statements with short response times.

The precompiler products offer two command line options, HOLD_CURSOR=YES and REBIND=NO, to hold cursor areas open and to retain the parsed and bound data. Consequently, SQL statements will not be reparsed or rebound even if the statements are executed many times.

For example, in one test with a transaction of about two dozen SQL statements, response time for the transaction was reduced from 20 seconds to 4 seconds by setting HOLD_CURSOR=YES and REBIND=NO.

20

DATABASE TUNING

A primary responsibility of the DBA is to "tune" the database system so as to provide the best possible overall performance for all users of a database. This chapter discusses actions that are usually the responsibility of the DBA but that will benefit all database users and applications.

This chapter describes basic tuning procedures and various parts of the ORACLE kernel and how to tune them. This chapter does not address the tuning of shared disk systems; for information on these systems see Chapter 21. Furthermore, many recommendations for system tuning are specific to an operating system environment; such recommendations can be found in the *Installation and User's Guides* for each operating system.

The ORACLE RDBMS supports DBAs as they both *tune* and *monitor* a system. A DBA may tune a database in many ways; for example, by adjusting numerous database structures or the variable parameters found in the INIT.ORA file. A DBA may monitor a database by regularly checking a number of statistics that are displayed using the SQL*DBA command MONITOR.

Basic Tuning Concepts

Tuning a database for optimal performance is a complex task that may take much time and research. Many factors contribute to the process; for example, database usage and conditions are rarely static, and may change widely over time. Characteristics of databases vary widely, as do expectations and requirements, making it impossible to propose guaranteed performance guidelines. Nevertheless, the guidelines and procedures for tuning an ORACLE database system can be generalized to some degree. This chapter attempts to provide that generalization and simplification of tuning concepts.

The architecture of ORACLE RDBMS Version 6 contains several features specifically designed for high performance, including:

- row level locking
- deferred writing.

Row level locking allows inserts and updates to the same table to occur concurrently, as long as there is no contention for individual rows.

Deferred writing means that only a logical representation of the change needs to be written to a log file on disk for the commit to complete, and writing the modified database block can be deferred. Several transactions can commit with a single write to the log file, if they are ready to commit at the same time.

The high concurrency of ORACLE Version 6 allows full utilization of the processing power available in multiprocessor machines. These machines typically deliver more CPU power per cost than uniprocessors. Since CPU power is the major determinant of transaction throughput in Version 6, multiprocessors benefit.

Two basic concepts are associated with tuning a database system:

- optimizing memory usage
- optimizing disk I/O.

In other words, you want to optimize both *processing and storage*. The goal is to tune a system so that you balance the load on memory (CP) and I/O equally. On an optimally tuned system, both the CPU and the disks operate at nearly 100 percent capacity.

Know Your Users' Requirements

A prerequisite for effectively tuning a system for good performance is a thorough knowledge of current and expected requirements to be placed on the system. For example, you will want to know such things as:

- On average, how many unique ORACLE usernames will be connected at any one time?
- What is the maximum number of ORACLE users who would ever need to be connected?
- How many users will be doing what activities? For example, will many users be using SQL*Forms with all others using Pro*FORTRAN? Or will all users use SQL*Plus for ad hoc querying?
- For applications in high use, how long are the average and longest transactions?
- What auditing information is required? is desirable?
- What types of SQL statements are typical in the high use applications? Do users query many or few tables? Do they require many temporary segments? Are many users creating tables or altering grants?
- How sophisticated are the data schema? Are tables normalized, or are they flat operating system files which have simply been converted into tables?
- Have indexes and clusters been chosen carefully?

How to Tune Effectively

In order to tune your database effectively, you must be able to measure performance. You can use several scales or units of measurement, but what is important is that you can satisfactorily establish that a given tuning change consistently results in a demonstrable improvement.

Thus, you must observe your chosen units of performance over time. You should note significant numbers or measurements when the system is unusually busy or slack, or when significant events are occurring. These ranges give you base performance numbers upon which you'll want to improve.

Levels of Performance Tuning

By taking a systematic approach to tuning you will see the best possible performance improvements in the shortest time.

Tuning performance may occur at many levels, many of which are beyond the scope of this chapter. Furthermore, some options are available only when installing or creating a database (before it is in

operation) and others can be made at any time during database operation.

SQL Statement Tuning entails using the SQL language optimally. Although you can often obtain the same results using more than one SQL statement, you may notice surprising performance differences depending on the actual syntax used. See Chapter 19 for more discussion of the impact of wording SQL statements.

Application Design Tuning includes tuning applications as implemented using tools such as SQL*Menu, SQL*Forms, and Pro* precompilers. This is a broad topic only tangentially covered in this chapter. You should also refer to the books describing each individual product.

Database Structure Tuning includes using normalization theory to optimally organize your data in database tables and clusters, and to create the best set of indexes on those structures. Although normalization theory and data schema and design are beyond the scope of this book, there are several excellent texts on this topic. Chapters 5 and 16 do discuss database structures and their use. Chapter 19 discusses the use of indexes.

ORACLE RDBMS Configuration Tuning consists primarily of properly adjusting the variable parameters in the INIT.ORA parameter file. This topic is covered in this chapter and Appendix D contains a thorough description of the INIT.ORA file and each parameter.

Operating System Tuning includes a number of specific steps that vary across operating systems. This chapter gives examples of this tuning; for details, see your *Installation and User's Guide*.

Hardware Tuning consists of making the best use of the hardware available to you (such as CPUs, disks, and communications equipment) in order to reduce I/O contention and balance loads. This topic is covered in this chapter but is also system dependent; for additional details, see your *Installation and User's Guide*.

Primary Tuning Features

The ORACLE RDBMS itself contains many features that may be tuned by the DBA. These features include:

- system configuration file (the INIT.ORA file)
- buffer manager
- DBWR background process
- LGWR background process
- rollback segments.

Tuning at Installation Time

Some steps to improve performance can be taken only at installation time. The *Installation and User's Guides* for the various operating systems describe these steps; some general information follows.

Examples of O/S Tuning Steps

Information likely to be described in the installation guides includes:

- suggested directory structures or disk locations for ORACLE files
- storage requirements and recommendations
- memory requirements and recommendations
- process privileges and priorities
- benefits of shared ORACLE code
- preferred storage devices (for example, for speed and capacity)
- DMA (direct memory access)
- system usage/load.

Setting Process Priorities

All ORACLE processes are assigned equal priority. Attempting to change a task's priority will not improve performance, and may have an adverse affect on the system. See your *Installation and User's Guide* for more information.

Avoiding Disk Fragmentation

The ORACLE RDBMS interacts best with the operating system if all database files are contiguous on disk. You can assure contiguous files by creating the database on an empty disk or by using a disk-keeper or disk-optimizer program (when the ORACLE RDBMS is not running).

Quick Tuning Techniques

This section lists some quick and easy tuning steps that can improve the performance of your system significantly. These steps are straightforward and do not require extensive knowledge of the ORACLE system and performance monitoring tools. See later sections in this chapter for more details on each step.

- Place redo log files on a separate disk from database file(s).
- Reduce the frequency of checkpoints by increasing the value for the INIT.ORA parameter LOG_CHECKPOINT_INTERVAL.
- Reduce the frequency of log allocations by increasing the value for the parameter LOG_ALLOCATION.
- Increase the size of the SGA by increasing the parameter DB_BLOCK_BUFFERS.
- Create new rollback segments in addition to the SYSTEM rollback segment.

Optimizing I/O

As the number of users simultaneously accessing a database increases, the importance of balancing I/O increases. Your goals with respect to I/O probably include:

- balancing three different types of I/O:
 - writing to the log file
 - reading from the database files
 - writing to the database files
- avoiding I/O bottlenecks
- minimizing I/O paths
- optimizing the movement of data from disk to SGA buffers and back to disk
- supporting rapid I/O for the log file.

Optimally you would identify all data that will be accessed simultaneously in order to place that data on separate disk drives. This would allow maximum parallel I/O and thus minimum chance for I/O contention.

You may affect I/O in several ways, including:

- experimenting with storage parameters for individual database objects
- optimizing disk usage by configuring data across many files located on different disks
- separating table data from index data using tablespaces
- optimizing the SGA.

Monitoring I/O with SQL*DBA

To see the number of physical reads and writes to disk on a per-file basis, use the MONITOR FILE command. Refer to Appendix B for an example of the MONITOR FILE display and how to interpret the data.

Using Default Tablespaces

If many users will be accessing a database concurrently, it may be beneficial to spread their activity across tablespaces. This assumes that the database has multiple tablespaces, and that the assignment of tablespaces is not done randomly, but based on a knowledge of each user's applications and data.

Examples of ways to assign tablespaces include:

- assigning a particular tablespace to be used primarily for temporary segments
- assigning all users a tablespace other than SYSTEM for the creation of their database objects
- grouping users of individual applications so that their data is either distributed across a few tablespaces or resides in one tablespace dedicated to that application.

The last example recommends two very different possibilities neither of which is always preferable, because other considerations, such as recovery and archival strategies, also affect which tablespace scheme is preferable.

Minimizing the Number of Extents

In most cases, the fewer the number of data and index extents per table or index, the better. With the exception of "table striping" (described later in this chapter), the RDBMS deals more efficiently with a table or index stored in a single extent than with one in multiple extents. The use of few extents helps to minimize the overhead required in order to access any given row.

In particular, full table scans benefit from a reduced number of extents. Full table scans perform slower against data in many extents as they must span discontiguous spaces and the table segment header must be re-accessed before each extent can be entered.

You can query the data dictionary view `USER_SEGMENTS` to see how many extents a table or index currently occupies.

To minimize the number of extents you can use one or more of the following methods:

- create tables with an `INITIAL` storage allocation that will contain the entire table. (especially for tables that will remain static in size)
- adjust the storage parameters for a table or index to use larger extents by increasing the `NEXT` or `PCTINCREASE` storage parameters
- reduce the frequency of additional extent allocations (by making each extent larger, as above)
- increase the number of rows per block by decreasing the storage parameter `PCTFREE`
- consolidate fragmented tables into one extent.

To consolidate fragmented tables, use the following steps:

1. Rename the original table.
2. Create a new table with appropriately adjusted storage parameters.
3. Copy the data into the new table and check it.
4. Drop the original table.
5. Recreate the indexes.

When More Extents are Desirable

Occasionally it may be desirable to spread data across multiple extents to reduce block-level contention and expedite many simultaneous updates. This is particularly true when a table is very heavily accessed and updated. In this case, it is preferable to have fewer rows per block, and thus, the total number of blocks may increase. (Although the number of blocks will increase, the number of extents may or may not

increase, depending on the storage parameters FIRST, NEXT, and PCTINCREASE.)

To reduce the amount of storage (and thus rows) allowed per block for a table, increase the value for the PCTFREE storage parameter in the CREATE TABLE statement. You can even store only one row per block by specifying a PCTFREE of 95 or even 99. (Be sure to lower the PCTUSED parameter accordingly.)

Note that an allocation of one row per block is not practical for most tables, particularly large tables. A high PCTFREE usually results in a lot of wasted space. Large tables will waste more space than small tables, and will reduce the amount of effective caching. Additionally, the performance of full table scans will suffer because of the increased number of blocks that must be retrieved.

Another case when more extents are desirable is when you have a "fragmented" database. In a fragmented database, the free storage is not one contiguous space, but is many small bits of free space throughout the database files. In this situation you may be forced to use more numerous, smaller extents, even if only temporarily until you can reorganize the database.

In the second case you are not interested in reducing the number of rows per block (and effecting an increase in the number of blocks); rather your intent is to set the storage parameters so that more extents (of fewer blocks each) are used to store the data.

Reducing Frequency of Extent Allocations

As a table grows in size it must allocate a new extent whenever it runs out of space. In rapidly growing tables these allocations may occur frequently. As extent allocations consume a significant amount of resource, it is preferable to reduce the frequency of these allocations.

To reduce the number of extent allocations, use one or more of the following methods:

- Anticipate the eventual size of the table and allocate an INITIAL extent that should be adequate for the ultimate size of the table.
- Specify a large value for the storage parameter NEXT so that the subsequent extents are large and thus infrequent.
- Use a non-zero value for the storage parameter PCTINCREASE so that each new extent is larger than the previous extent.

Separating Table Data from Index Data

The placement of table and index data within the database can have a large affect on performance, and is easy to control or change. You should create tables and indexes that are frequently accessed on numerous separate disks. This allows user processes to read the database in parallel with each other and with DBWR.

You determine the location of tables and indexes when you create them. The CREATE TABLE or CREATE INDEX statements should refer to various tablespace names (where the tablespaces are comprised of files created on different disks).

To change the tablespace for a particular table, you can create a new table in the tablespace that loads all the table's data, drop the old table, and rename the new table with the old name.

To change the tablespace for an index, simply drop it and recreate it in the desired tablespace.

"Striping" Large Tables

Striping consists of breaking a large table into sections and spreading the sections across different disks to allow for simultaneous access by many users. For example, to divide a 100 Mbyte table into 10 sections:

1. Create a tablespace consisting of 10 files, each slightly larger than 10 Mbytes in size (to allow for some overhead). Create each of the 10 files *on a separate disk*.

2. Create the table using these storage parameters:

```
storage(initial 10m next 10m minextents 10 pctincrease 0)
```

3. When the table is loaded, the data will be spread across all ten files.
All of the data need not be loaded at the same time.

Database and Control File I/O

A database system requires a minimum of one database file, one control file, and two redo log files, in addition to the ORACLE program files. Optimal file placement affects both RDBMS performance and recovery strategies in case of media failure.

Regarding the placement of database and control files, you should:

- place all database files on separate disks to avoid disk contention and increase parallel access
- place multiple copies of control files on multiple disks.

Log File I/O

As the redo log files experience heavy I/O, it is strongly recommended that you place the redo log files on a disk separate from the database files. This allows the background process LGWR to write in parallel with the rest of the system.

Because log file I/O must occur to support committed transactions, you should try to support rapid log file I/O. The background process LGWR is the only process to write to the log files, and it writes asynchronously. The most important step for supporting log file I/O is to place redo log files on a separate disk from all database files. All redo log files may be on the same disk as each other, as they are always written sequentially by LGWR, never concurrently.

Optimizing the Buffer Manager

The following description applies to ORACLE versions 6.0.26 and earlier. For a description of version 6.0.27 and later, please see the *ORACLE RDBMS Performance Tuning Guide*.

Two important factors of performance are the size of the buffer cache in the SGA and the efficiency of the background writer process DBWR. The *buffer cache* (also called the *buffer pool*) contains copies of recently modified database blocks; each buffer corresponds to a modified database block. DBWR is responsible for writing the buffers from the cache to disk.

Because the writing of modified buffers is deferred until after commit, more modified buffers are likely to be in the cache at any one time. To support increased concurrency, it is important to identify which buffers should remain in the cache and which should be written to disk. Thus, tuning of the buffer manager, the buffer cache, and the DBWR process can have a significant impact on performance.

When a user process needs to access a database block for any SQL operation, the process first checks to see if the block is already in the buffer cache. This search uses a hashing scheme; if the desired block exists in a buffer the user process quickly locates it. If the block is not in the cache, then the user process must find a *free buffer* (that is, one that has no modified data in it) by searching through the LRU list of buffers.

The ORACLE buffer manager maintains the LRU list of buffers (LRU stands for Least Recently Used). This list orders each buffer based on how recently its data was last accessed. The top of the LRU list contains the most recently used buffer and the bottom contains the least recently used buffer. The buffer manager ensures that more frequently accessed data is cached longer than less frequently accessed data.

When the user process finds a free buffer (whether in the cache or in the LRU list), it reads the database block into that buffer and then moves that buffer to the top of the LRU list.

When a buffer is not used for a long time it migrates to the bottom of the LRU list and is eventually "cleaned" by DBWR and reused as a free buffer. A buffer containing modified data cannot be declared "free" until its contents have been copied to disk by DBWR.

Monitoring the Buffer

To monitor the activity of the buffer manager, use the SQL*DBA MONITOR command with the STATISTICS option. The following statistics relate to the buffer manager:

- physical reads
- physical writes
- buffer busy waits.

Increasing the Number of Database Buffers

The most important variable INIT.ORA parameter you can set is DB_BLOCK_BUFFERS. This parameter controls the number of buffers in the database buffer pool.

The default value of 32 buffers is too low for any realistically sized database. A more realistic value is 100 or 200. If your database storage requires more than 100 megabytes, you may consider setting this parameter to 1000 or higher. Buffer pools of up to 20Mb have been configured on large VAX/VMS systems (64Mb memory). The optimal size however, is dependent upon your applications and operating system.

By increasing the value for DB_BLOCK_BUFFERS, you also increase:

- the size of the SGA (the buffer pool)
- the amount of data that can be cached
- the probability that data will remain cached
- memory usage
- paging.

The net effect is to improve performance, as long as paging is not excessive. The only tradeoff of increasing the size of the buffer pool is that memory usage and thus paging of both the SGA and the user processes will increase. Paging can be reduced or eliminated by proper tuning of your operating system (see your *Installation and User's Guide*).

Reducing Block Level Contention

Even a row-level locking system may experience decreased performance when several users update different rows in the same block. Although several modified rows from different transactions may reside in the same block at the same time, the entire block is locked during the instant a user is actually modifying the block. As the actual writing time is very small, the chance of two users trying to lock the same block is minimal; however on heavily accessed blocks this will happen. The statistic *buffer busy waits* records "block-collisions" of this type.

If you think that block collisions in heavily-used tables may be degrading performance, then you might reduce the number of rows per block in these tables (see "When More Extents are Desirable" earlier in this chapter). By so doing you reduce the likelihood that different transactions will access different rows in the same block.

Tuning the DBWR Process

The following description applies to ORACLE versions 6.0.26 and earlier. For a description of version 6.0.27 and later, please see the *ORACLE RDBMS Performance Tuning Guide*.

If your INIT.ORA parameter DB_BLOCK_BUFFERS is set to 500 buffers or less, then the default INIT.ORA parameters will probably provide satisfactory performance. However, if you have set DB_BLOCK_BUFFERS higher than 500, then you should tune your system to reduce the amount of work for the DBWR process and the user processes.

The following INIT.ORA parameters are most useful in tuning DBWR:

- DB_BLOCK_MAX_MOD_PCT
- DB_BLOCK_MAX_CLEAN_PCT
- DB_BLOCK_TIMEOUT_WRITE_PCT

Each of these parameters is specified as a percentage of the number of DB_BLOCK_BUFFERS.

How DBWR Works

As a user process searches the LRU list for a free buffer, it counts the ineligible buffers as it passes over them. If the user process passes over DB_BLOCK_MAX_MOD_PCT percent of the buffer pool when searching for a free buffer, it will notify the background process DBWR to begin cleaning buffers. A user process stops looking for a free buffer when it has read DB_BLOCK_MAX_SCAN_PCT buffers; if it has not

found a free buffer in this case, the statistic *dbwr free needed* is increased, while the user process waits for DBWR to clean a buffer for its use).

When notified, DBWR starts at the bottom of the LRU list and begins cleaning the least recently used buffers. If a buffer has any modified data in it, DBWR "cleans" it by copying its contents to disk if necessary and marking it as a free buffer. DBWR continues cleaning buffers until DB_BLOCK_MAX_CLEAN_PCT of the buffer pool is clean.

When the database is idle, DBWR wakes up periodically (at "timeouts") and cleans as many buffers as are specified by DB_BLOCK_TIMEOUT_WRITE_PCT. If the database is idle long enough, DBWR will eventually clean the entire buffer pool.

Thus, DBWR is notified in two circumstance:

- when a user process had to read more than DB_BLOCK_MAX_MOD_PCT buffers in order to find a clean buffer
- periodically, for timeouts.

Monitoring DBWR

To monitor the DBWR process use the SQL*DBA MONITOR STATISTICS. You should try to keep the following statistics low:

- DBWR buffers scanned
- DBWR free low
- DBWR free needed
- DBWR timeouts
- free buffer inspected
- free buffer waits.

You should also monitor the following latches, using the SQL*DBA command MONITOR LATCH:

- cache buffer chains
- cache buffers LRU chain.

For these two latches, the number in the TIMEOUTS column should be kept to a minimum.

Minimizing DBWR Activity

Ideally, DBWR should work just hard enough to provide an adequate supply of free buffers so that database read operations are not constrained.

If DBWR does not work hard enough, the system will run out of free buffers and users must wait for a buffer to be cleaned. The statistic

dbwr free needed indicates how many times the system ran out of free buffers; this statistic should be kept to a minimum.

If DBWR works too hard, it will write out buffers that may be changed again in the near future and will only have to be requested again. In this case, user processes will wait for DBWR to finish cleaning buffers and the statistic *free buffer waits* will be high.

During its cleaning, DBWR acquires and releases locks on the buffers and portions of the LRU list. This locking may contend with user processes if DBWR is very active. Therefore, the number of buffers DBWR cleans in each pass should be reduced to a minimum. The INIT.ORA parameter DB_BLOCK_MAX_CLEAN_PCT determines the maximums number of buffers that DBWR will clean; the actual number of buffers is indicated by statistic *DBWR buffers scanned*.

If the buffer pool contains many modified buffers, a user process must search longer to find a free buffer. The number of buffers the user process needs to search should be minimized as well. The maximum number of buffers a user process will search is set by the INIT.ORA parameter DB_BLOCK_MAX_SCAN_PCT; the actual number of buffers searched is indicated by the statistic *free buffers inspected*.

Lowering the value for INIT.ORA parameter DB_BLOCK_MAX_MOD_PCT will reduce the statistic *free buffer inspected*, while raising it will lower the statistic *DBWR buffers scanned*. Because it is desirable to have both of these statistics low, you may have to try a few values for DB_BLOCK_MAX_MOD_PCT to find the optimal value.

The values for the two statistics *DBWR free needed* and *free buffer waits* should always be zero. If they are not, then increase the value for INIT.ORA parameter DB_BLOCK_MAX_CLEAN_PCT.

Tuning the Redo Log

The redo log contains the changes made to database blocks. If the system crashes or a disk fails, the redo log is used to restore the changes made to the database since the last checkpoint.

Like the database buffer pool, a *redo buffer pool* exists in the SGA. Every time a user process modifies data in the database buffer pool, it makes a similar write to the redo buffer pool. The redo buffer pool is organized in a sequential manner. Rather than holding database blocks, it holds modified data *only*. When each user writes modified data to the redo buffers, it begins where the last user left off. For a commit to occur, only the data that the user wrote to the redo buffers needs to go to disk.

The background process LGWR is notified in three conditions:

- whenever a commit is requested
- whenever DBWR needs to clean the buffer blocks
- whenever the redo log buffer is full.

When notified, LGWR writes any modified redo buffers to disk (whether committed and non-committed). The commits of many users can be bundled together in a single I/O (called "piggy-backing"). This size is determined by the value for the INIT.ORA parameter LOG_IO_SIZE.

The more commits that are bundled together (as measured by the "batching factor"), the better the performance. The batching factor is the average number of transactions per write, as measured by the following ratio:

$$(\text{user commits} / \text{small redo writes}) = \text{transactions} / \text{write}$$

Summary of Redo Log Optimizations

The following list summarizes steps for optimizing the redo log:

- Place online redo log files on fast devices.
- Use separate disks from database files.
- Increase the value for INIT.ORA parameter LOG_BUFFER to reduce log file I/O
- Reduce checkpoint frequency by increasing the value for parameter LOG_CHECKPOINT_INTERVAL.
- Reduce log allocation frequency by increasing the value for the parameter LOG_ALLOCATION.

Monitoring the Redo Log

To monitor the redo log, use the SQL*DBA command MONITOR STATISTICS. The statistics of most interest regarding redo log activity are:

- DBWR checkpoints
- chunk allocations
- small redo writes
- large redo writes

Optimizing the Log Buffer

The INIT.ORA parameter LOG_BUFFER determines the size in bytes of the log buffers cached in the SGA. Higher values reduce redo log file I/O, particularly if there are many long transactions.

Reducing Checkpoint Frequency

Checkpoints can momentarily decrease performance of the database. Checkpoints occur when a certain number of redo log file blocks have been written (as specified by the value for the INIT.ORA parameter LOG_CHECKPOINT_INTERVAL) and whenever an online redo log file fills (at log switch).

During a checkpoint, DBWR insures that all blocks modified since the last checkpoint are actually written to disk. Since DBWR has been writing database buffers continually since the last checkpoint, this does not necessarily mean that many database buffers will be written at once. The potentially high amount of DBWR activity may cause some contention in the system, and thus it is desirable to keep the number of checkpoints to a minimum.

To maximize the time between checkpoints you can:

- Create large redo log files, of 10 megabytes or more if possible.
- Set the parameter LOG_CHECKPOINT_INTERVAL to a very high number (like 500000) to ensure that a checkpoint will only occur when you switch log files.

The default value for LOG_CHECKPOINT_INTERVAL is low enough so that several checkpoints will usually occur while a single log file is in use.

The log file switch and associated checkpoint are inevitable, but can be reduced to a minimum by creating large log files. On a heavily used system, log file sizes of 5 to 10 megabytes each should be sufficient to allow at least a few minutes between checkpoints. Ultimately, the frequency of checkpoints depends on the amount of user activity.

Note: An increase in time between checkpoints will also increase recovery time after a system crash.

To see how many checkpoints have actually occurred for a given instance, check the statistic *dbwr checkpoints*.

Reducing Log Allocation Frequency

A log allocation occurs whenever an instance needs more space in an online redo log file to write to.

In single instance systems it is not necessary to allocate the redo log among instances, so you should set the INIT.ORA parameter LOG_ALLOCATION to the size of your largest redo log file. (This means increasing the default value, which is set so that several log allocations will take place within each log file). Setting a high value ensures a single allocation per log file and reduces the occurrence of log allocations (which are expensive) to one per log file, or one per checkpoint.

To see how many log allocations have actually occurred for a given instance, check the statistic *chunk allocations*.

Shared disk systems must use values for LOG_ALLOCATION that reflect the number of instances and the sizes of the redo logs, keeping performance in mind. See Chapter 21 for more information on setting this parameter in shared disk systems.

The Interaction between Parameters and Statistics

The following tables summarize the relationships between significant statistics and latches and INIT.ORA parameters for ORACLE versions 6.0.26 and earlier. For version 6.0.27 and later, please refer to the *ORACLE RDBMS Performance Tuning Guide*.

Statistic	Desired Change	Recommended Actions
chunk allocations	reduce	Increase LOG_ALLOCATION
buffer busy waits	reduce	Increase number of rollback segments and/or change smaller tables in the database to have one row per block.
DBWR buffers scanned	reduce	Decrease DB_BLOCK_MAX_CLEAN_PCT and/or DB_BLOCK_TIMEOUT_WRITE_PCT
DBWR checkpoints	reduce	Increase LOG_CHECKPOINT_INTERVAL and/or increase size of the redo log files.
DBWR free low	reduce	Increase LOG_CHECKPOINT_INTERVAL and/or increase DB_BLOCK_MAX_CLEAN_PCT
DBWR free needed	reduce	Increase DB_BLOCK_MAX_SCAN_PCT and/or increase DB_BLOCK_MAX_CLEAN_PCT
DBWR timeouts	reduce	Decrease DB_BLOCK_MAX_MOD_PCT
free buffers inspected	reduce	Increase DB_BLOCK_MAX_CLEAN_PCT and/or increase DB_BLOCK_TIMEOUT_WRITE_PCT
free buffer requested	reduce	Increase DB_BLOCK_BUFFERS and/or tune the application to make more efficient use of the cache.

<i>Statistic</i>	<i>Desired Change</i>	<i>Recommended Actions</i>
free buffer scans		Increase DB_BLOCK_BUFFERS and/or increase DB_BLOCK_MAX_CLEAN_PCT and/or increase DB_BLOCK_TIMEOUT_WRITE_PCT
free buffer waits	reduce	Increase DB_BLOCK_BUFFERS and/or increase DB_BLOCK_MAX_CLEAN_PCT and/or increase DB_BLOCK_TIMEOUT_WRITE_PCT and/or raise the priority of the DBWR process.
physical reads	reduce	Increase DB_BLOCK_BUFFERS
physical writes	reduce	Increase DB_BLOCK_BUFFERS
(User commits/small redo writes)	increase	Increase LOG_BUFFER and/or increase LOG_IO_SIZE.
<i>Latch</i>	<i>Desired Change</i>	<i>Recommended Actions</i>
cache buffer handles		Decrease DB_BLOCK_MAX_CLEAN_PCT and/or decrease DB_BLOCK_TIMEOUT_WRITE_PCT
cache buffer chains		
cache buffers LRU chain		

Tuning Rollback Segments

Rollback segments contain the information necessary to rollback uncommitted transactions. Before a user process modifies a database block, it first writes the inverse of that modification to a rollback segment. The affected database block immediately reflects the modification whether or not the transaction is committed, but if the transaction is cancelled (by an explicit or implicit rollback), the information in the rollback segment is used to restore the database block to its pre-transaction state.

Each time a user process starts an update the process is assigned a rollback segment to use for that transaction. For the duration of the transaction, the user process writes rollback information only to that segment.

When a transaction is committed, the rollback information is released but is not immediately destroyed. The information will remain in the rollback segment to be used by concurrent queries. Rollback segments are written sequentially and wrap around to ensure that rollback data exists for as long as possible.

If a transaction is so long that it fills up the current rollback segment extent with uncommitted data, then a new extent must be allocated and appended to the rollback segment.

The primary INIT.ORA parameters of use in tuning rollback segments are:

- ROLLBACK_SEGMENTS
- TRANSACTIONS_PER_ROLLBACK_SEGMENT

Monitoring Rollback Segments

To see the current status of rollback segments, query the view DBA_ROLLBACK_SEGS. Rollback segments are marked either "INUSE" or "AVAILABLE". You can also use MONITOR ROLLBACK to see rollback segment statistics.

Read Consistency

The read consistency model guarantees that the results retrieved for every query must be correct as of the time the query was initiated.

A user process that is executing a multi-row query may request rows that were modified by another user after the query was initiated. In this case, the query uses the rollback information associated with that row to reconstruct the data for that row as of the time the query started. If more than one change had been made to that row, then the query process must go through several iterations to reconstruct the proper version of the row.

Note: All row reconstruction takes place within the user's private temporary space. Thus there may be many read consistent versions of any row, depending on the frequency of access.

Using Multiple Rollback Segments

When a transaction writes to a rollback segment, it locks the segment header during the time it is actually doing the writing. Generally, the writes take place quickly and the header is locked very briefly. However, when there are many concurrent transactions, there may be contention and users will queue on the header block. The statistic *header units/sec* in the MONITOR ROLLBACK screen indicates how often this condition occurs. The best solution is to create and use more than one rollback segment.

Because rollback segments are associated with an instance on start up, all transactions originating from that instance are distributed across all rollback segments assigned to that instance. Thus, the addition of each rollback segment for an instance reduces contention.

By default, only the SYSTEM rollback segment exists when a database is first created. The kernel and background processes use this segment for housekeeping. To add more rollback segments, refer to Chapter 16.

Rollback segments may be sized and located so as to aid load balancing and processing efficiency. For example, a segment may be located on a particular local disk or it may be explicitly sized for certain applications on a particular instance.

Choosing a Size for Rollback Segments

Smaller rollback segments are more likely to remain in the cache longer. However, larger rollback segments are better for concurrent queries. Long-running concurrent queries may need to look through very old rollback information to construct read-consistent data. To support long running queries, you should increase the size of your rollback segments.

If more than one rollback segment is active, and several transactions occur simultaneously, then ORACLE assigns the transactions to the rollback segments as evenly as possible among the segments.

If all rollback segments are the same size, they will all fill up at the same rate, and will wrap around at roughly the same time. If one segment is smaller than the rest, then it will wrap first and overwrite old rollback information, even though there is still unused space in the larger rollback segments.

Because concurrent queries are constrained by the size of the smallest rollback segment, it is recommended that all rollback segments be the same size.

Caching Rollback Segments

Like database blocks, rollback segment blocks reside on disk or in the database buffer pool, and are subject to the LRU scheme of the buffer manager. If the rollback segments are small enough or the buffer pool large enough, the rollback segments are more likely to be cached. This will improve performance in systems characterized by numerous short queries.

Thus, rollback segments should be as small as possible, while still maintaining enough space to accommodate read-consistent queries.

Monitor Ongoing Database Storage and Use

Choosing Reasonable Auditing Options

If you enable auditing, both processing and storage overhead result. After auditing activity is enabled with the INIT.ORA parameter AUDIT_TRAIL, users can request auditing with SQL statements. Because it is easy to request auditing, and because auditing information can accumulate rapidly, DBA's should encourage users to carefully judge what auditing information is truly useful.

Problem Determination

If you believe your system is encountering bottlenecks, your system is:

- CPU-bound
- latch-bound
- DBWR-bound
- LGWR-bound
- I/O bound

A system is "bound" by a resource (such as DBWR or I/O) if that resource is slowing or restricting the performance and if by expanding that resource the performance would increase. In other words, that particular resource has reached its limit, can do no additional work any faster, and is thus a constraint on the entire system.

CPU-Bound

The theoretical maximum throughput of ORACLE Version 6 is usually attained when a system is CPU-bound. To tell if your system is CPU-bound requires operating system specific diagnostics and corrective actions.

When a system is CPU bound, each percent decrease in CPU time per transaction usually translates directly into a percent increase in throughput.

Latch-Bound

Your system may be latch-bound if the number of timeouts is high. Latches are the low level locking mechanism used to serialize access to data structures in the SGA. Latches are held for very short durations and are never held across I/O or timeouts.

Latches impose a theoretical maximum throughput on a system. To monitor latches, use the SQL*DBA command MONITOR LATCHES; however, the interpretation of the data is operating system specific, as

are the corrective or preventive measures. See your *Installation and User's Guide* for more information.

DBWR-Bound

A system that cannot do I/O fast enough may be DBWR-bound. To see whether this is a problem, check the I/O rate in the MONITOR STATISTICS display. If it is a problem, you have several options, discussed earlier, for tuning the DBWR background process.

LGWR-Bound

To see whether your system is LGWR-bound, measure the I/O rate on the device that contains the redo log files. Observe the I/O rate on the drive corresponding with the transaction rate, until the transaction rate exceeds the number of I/Os that the drive can handle synchronously. At this point the I/Os will be pegged but the transaction rate can continue to increase by increasing the batching factor (transactions per physical write).

I/O Bound

A system can become I/O bound in three different ways:

- Redo log file I/O can become a bottleneck if the amount of changed data generated per unit of time exceeds the throughput of the disk subsystems.
- Database reading can become a bottleneck if the number of pages required from a single disk or controller is greater than the capacity of that component.
- Database writing, done by the background process DBWR, can become a bottleneck if DBWR cannot keep up with the number of modified database buffers.

21

SHARED DISK DATABASE SYSTEMS

Two can live as cheaply as one — if they both have good jobs.

Sigmund Freud

This chapter addresses the special DBA concerns regarding administering a database which is shared by multiple CPUs or instances. Information in this chapter complements that found in the rest of this guide; it simply includes discussions specific to the sharing of a common database by multiple instances.

What is a Shared Disk System?

When a database is accessed by multiple instances simultaneously, it is called a *shared disk system*. Typically, the database is on shared disk drives while the instances accessing the database are on distinct CPUs which share those disks. Refer to figures in Chapter 1.

Using SQL*DBA for Multiple Instances

SQL*DBA commands normally apply to the default instance, which is the local host machine. If SQL*Net is available, the DBA can connect to other instances using the SQL*DBA command SET INSTANCE (see Appendix B).

Each instance should have its own individual INIT.ORA file. Each of those INIT.ORA files should name the same control file(s) for the parameter CONTROL_FILES. Thus, the INIT.ORA file is not shared but the control files are shared by the instances.

Startup with SHARED Option

For shared disk systems, such as VAXClusters, each instance must mount the database in SHARED mode, or STARTUP the instance using the SHARED option:

```
SQLDBA> STARTUP SHARED
```

If the option is omitted, EXCLUSIVE is the default. If any instance opens a database without the SHARED option, no other instance can open the database.

If one instance has opened a database with the SHARED option, then every other instance opening the database must also use the SHARED option.

INIT.ORA Parameters Relevant to Shared Disk Systems

The parameters with the prefix "GC" are relevant primarily for shared disk systems:

- GC_DB_LOCKS
- GC_ROLLBACK_LOCKS
- GC_ROLLBACK_SEGMENTS
- GC_SAVE_ROLLBACK_LOCKS
- GC_SEGMENTS
- GC_SORT_LOCKS
- GC_TABLESPACES.

The prefix "GC" stands for Global Cache. Their settings determine the size of the global collection of locks which protect the database buffers on all instances. The settings you choose will have an effect on the use of certain operating system resources.

For additional information on setting these parameters refer to Appendix D and the *Installation and User's Guide* for your operating system.

Other parameters relevant to shared disk systems are:

- INSTANCES
- LOG_ALLOCATION

Using Rollback Segments

A rollback segment other than the SYSTEM rollback segment must be dedicated to each instance accessing a database. Rollback segments are not shared among instances. The rollback segment may be created by a DBA for any instance, and may be in any tablespace. An instance cannot start unless it has access to at least one rollback segment, whether it is public or private.

There are no performance differences between PUBLIC and private rollback segments.

Private Rollback Segments

To allocate a private rollback segment to one instance only, the DBA for the instance should:

1. Create the rollback segment with the SQL statement CREATE ROLLBACK SEGMENT.
2. Name the rollback segment in the INIT.ORA file as a value for the parameter ROLLBACK_SEGMENTS, thus reserving the rollback segment for that instance.
3. Stop and start the instance, to claim the new rollback segment.

A rollback segment should be named in only one INIT.ORA file, so it is associated with only one instance.

Public Rollback Segments

PUBLIC segments can be created by any instance and are available to any instance once created. Public rollback segments not currently being used can be "claimed" by an instance, which then will be the sole user of the segment until the instance shuts down, releasing the segment for use by another instance. A public segment is not usually named in any INIT.ORA files.

Backup and Recovery for Shared Disk Systems

Using the Redo Log

A database uses one redo log no matter how many instances share the database. If several instances share the database, they share each online redo log file as it is being written.

Instances sharing a database must write to the *same* online redo log file. If one instance runs out of space in a log file, then *all* instances must begin writing to the next log file.

Allocating Log Space to Instances

In a shared disk system, each instance sharing the database allocates a range of the current online redo log file. This allows more efficient use of the redo log file. These ranges are called *allocations* and their sizes are determined by the value for the INIT.ORA parameter LOG_ALLOCATION for each instance. When one allocation runs out, the instance allocates another allocation, either in the current or the next log file.

The size of each online redo log file should be at least as big as the sum of all LOG_ALLOCATION parameters for all instances sharing the database. This assures that each instance can allocate space in the online redo log file if all instances were to start simultaneously, and prevents unnecessary log switches. For example, if you have three

instances, and each has a LOG_ALLOCATION of about half of a redo log file, as soon as the third instance starts to write log entries a log switch is required, even if the other two instances have only written a little.

Following are some recommendations for setting up the log files:

- make LOG_ALLOCATION at least 1000 blocks
- make each log file at least 2000 blocks times the number of instances
- if possible, allow 4 to 5 allocations per file per instance.

Smaller allocation sizes make somewhat better use of log file space, because they leave less unused space when switching from one log file to another.

Checkpoints

Checkpoints are performed on a per instance basis. Specifically, a checkpoint will occur for each instance accessing an online redo log file under either of two circumstances:

- when the number of redo log blocks fills as indicated by the instance's own INIT.ORA parameter LOG_CHECKPOINT_INTERVAL
- when the current online log file fills (at log switch time).

The instances accessing the redo log can have different checkpoint intervals; checkpoints would then be taken for each instance when that number of blocks had been written, and a checkpoint occurs for all instances whenever an online log file fills.

Setting the Log's Mode

The mode of using the log (ARCHIVELOG or NOARCHIVELOG) is set at database creation, and can be changed via the command ALTER DATABASE. This characteristic is associated with the database rather than the individual instance, and thus all instances may share the responsibility of archiving log files. All instances should use the same mode and archive destination.

Archiving

Archiving is done on an instance basis. Archiving can occur from any or all instances, and can be either automatic or manual. Instances need not all use the same archiving method.

If multiple instances are each automatically archiving, then the first instance to notice a full log file will archive it. Only if all instances are using manual archiving will archiving remain a totally manual process.

You may prefer to have the instance with easiest access to a tape drive do the automatic archiving, while other instances do manual archiving, in the expectation that the one instance will do the archiving. If the only instance doing automatic archiving shuts down and the last online redo log file fills, then logging cannot continue, and the database system will stop until the log is archived and available for reuse.

Instance Recovery

For shared disk systems, instance recovery can be done automatically, because several instances are coordinating. If one instance fails, one of the other instances will notice, and perform instance recovery for the failed instance. Any instance can be the first to notice and perform recovery. Recovery does not include restarting the instance.

Occasionally an instance may fail in a way that is not recognized by other instances, and therefore no other instance will know to perform recovery on the failed instance. If this happens, the DBA should identify the failure, and shutdown that instance to perform recovery.

22

DISTRIBUTED DATABASES AND DISTRIBUTED PROCESSING

A community is like a ship; everyone ought to be prepared to take the helm.
Henrik Ibsen

This chapter describes special concerns related to databases spanning multiple computers or multiple databases and applications tools connected on a network. Topics include:

- a definition of terms and introduction to concepts
- identifying the components of a distributed system
- assuring data security and integrity.

Information in this chapter is intended to supplement that found in other Oracle Corporation documentation, particularly the SQL*Net books (for example, the general SQL*Net guides, manuals describing the individual communications protocols, and the *Installation and User's Guides*).

What is SQL*Star?

An ORACLE database can be used in a software configuration known as SQL*Star. SQL*Star is a collection of software, installed to support distributed processing or a distributed database, which usually includes the following three ORACLE components:

ORACLE RDBMS The ORACLE RDBMS inherently contains the functionality to support distributed databases (for example, to support distributed queries), location transparency, and site autonomy. These concepts are further described in this chapter.

SQL*Connect This product is a "gateway" product to non-ORACLE databases, in particular SQL/DS and DB2. SQL*Connect permits some Oracle Corporation tools to operate on the non-ORACLE databases as if they were ORACLE databases. In conjunction with SQL*Net, SQL*Connect can be used on remote databases.

SQL*Net SQL*Net is the heterogenous network interface component of SQL*Star. It allows data to be sent across communications protocols and handles the transfer of data between various databases.

Refer to the aforementioned SQL*Net documentation and to SQL*Connect documentation for details on those products. The emphasis of this chapter is the role of the DBA in maintaining a local database which is a part of SQL*Star.

What is Distributed Processing?

Distributed processing occurs when an application on one CPU accesses a database on another CPU. Some of the benefits of distributed processing include:

- You can use the most appropriate hardware for the job at hand. For example, personal computers (PCs) can be used for fast cheap processing, while mini or mainframe computers, with fast I/O and cheap and virtually unlimited disk space, can store the data centrally, for access by the PCs.
- You can support more users with cheaper hardware. More processing can occur on PCs, and capacity can be increased by buying PCs rather than hardware for the mainframe.

- Both departmental processing and central administration are possible. Shared data can be administered centrally by a DBA, while applications and processing tools can be controlled locally by users most concerned.

What is a Distributed Database?

A *distributed database* is a set of databases stored on more than one CPU in such a way that users perceive the data as a single "large" database, when in fact it is several "smaller" databases. Each local database is controlled locally by its DBA (maintained, started, backed up, and so on).

It is important to note that each CPU runs the software to access the other databases; there is no central software which coordinates all participating databases. This provides the advantages of site autonomy and avoids a potential central point of failure.

A *distributed database system* includes a distributed database and application tools that may be located on CPUs separate from the databases. It differs slightly from a distributed database because applications are on different CPUs than the data they require. Thus networking must be used for an application to access data.

Benefits of ORACLE Distributed Processing

In addition to the general advantages and flexibility that distributed databases and distributed processing offer, the joint use of ORACLE and SQL*Net offer two significant benefits: location transparency and site autonomy.

Location Transparency

In a distributed database data is spread across several "locations" (databases). A user may be interested in a table's location for two reasons:

- Data may be partitioned among several databases (e.g., LONDON and BOSTON) and the user may want table EMP from the LONDON database rather than the BOSTON database
- The data may reside on a single node of the distributed database and the user needs to know the node in order to access the table.

Location transparency resolves the second situation; it allows you to refer to a database object (such as a table or view) without regard to where it is. The database hides the location from you, requiring you to know only the object's name.

In addition, you can reference data on multiple nodes in a single statement. ORACLE and SQL*Net automatically and transparently route (parts of) SQL statements to remote nodes for execution if needed.

Some benefits of location transparency include:

- You need not know or remember where data is located.
- Objects can be moved with no impact on users or applications.

Note that location transparency is a characteristic of a distributed database, but not necessarily distributed processing. That is, in a distributed database you can query three tables and not know or care which databases contain the tables. In contrast, with distributed processing, you or your application may need to explicitly request a network connection to a specific database at a named node.

Site Autonomy

Site autonomy means that each database participating in a distributed database is administered separately and independently from the other databases, as though it were a non-networked database.

The benefits of site autonomy include:

- better failure resistance
- the database is available as long as one database and the network are available
- no central point of failure that could stop all operations
- each node continues operations if the network is down
- failure recovery is on individual site basis
- local control
- a data dictionary for each local database
- sites may upgrade software independently
- changes to one database affect only that database and need only be validated by that database
- the DBA's domain of responsibility is smaller and easier to control.

In an ORACLE distributed database, the participating databases have "peer to peer" relationships; each database is equal and has equal responsibilities.

The single exception to this is a node that is a single-process node such as a PC, whether it contains a database, an application tool, or both. Because only one user can access a database on a PC, PCs are not usually equal machines to mainframe or mini computers.

The Role of Clients and Servers

A distributed database system is made up of clients and servers.

A *client* is an application requesting data from another node. It may be a node with an application tool only, or it may be a node with a database. Clients can be PCs, minicomputers, or mainframes.

A *server* is any node with a database, of which data may be requested. Servers must be multi-user operating systems, running multi-user ORACLE.

One node can have many transactions occurring at one time; it may be a client in some transactions and a server in others.

Connecting Clients and Servers

A distributed database system requires that predefined pathways between participating databases be set up. These predefined links are called *database links* (DBLINKS) and are described in more detail later in this chapter.

For example, if you connect to Database-A, query against Table-M, and Database-A determines that Table-M is in Database-B, then Database-A will submit your query for you to Database-B, so long as it has a valid way to connect to Database-B. In fact, Database-A actually connects to Database-B on your behalf, creating a process there to execute your query, before returning the results back to Database-A.

Division of Labor between Clients and Servers

The client is responsible for executing the application. It sends SQL statements through the network to the server. The server parses and executes the SQL statements in order to send data and status values back to the client. Parsing of SQL statements cannot be done on the client machine, since the definition of the table is not present and the software required to parse (the ORACLE kernel and data dictionary) is not guaranteed to be present.

Connecting between Versions

Version 6 clients can access Version 5 server databases; however, they cannot use features unique to Version 6 since the server does not support them. Version 5 clients can access Version 6 servers.

Features of ORACLE and SQL*Net

ORACLE configured with SQL*Net supports *distributed queries*. This allows you to:

- reference data on nodes other than where you are logged in
- issue queries referencing data on single or multiple nodes
- perform joins and nested queries on tables on different nodes
- define views using tables on different nodes.

ORACLE and SQL*Net do not currently provide distributed update capability. Updates (UPDATEs, DELETEs, INSERTs, and also including DDL commands) can only be performed at the node you are connected to. Since you can only update data at the current logged-in node, the DBA should try to assure that data is located locally for its heaviest users.

A DBA should be assigned responsibility for each database participating in a distributed database, just as though it were an independent database. In fact, one DBA can be responsible for all databases or many DBAs may be; if there are multiple DBAs, they will need to coordinate with each other on some matters (primarily network issues) and it will be to their benefit if they also coordinate on table design, user access, and performance.

When multiple CPUs are connected by a network with the intention of allowing users and applications access to multiple nodes, several new DBA concerns arise. These concerns can be classified in the following categories:

- installation decisions (where to install which software and where to locate which data)
- guaranteeing data security and integrity across the network
- guaranteeing good system performance on all nodes and across the network.

If the database is a distributed database, then the potential for multiple applications accessing data across multiple CPUs is high. Thus, the DBAs have much flexibility in configuring and maintaining their databases and applications. This flexibility also implies more decisions, more tradeoffs, and perhaps some increased administrative costs. This chapter cannot address all circumstances and conditions that should be considered when designing a distributed database system, but it attempts to point out basic considerations for DBAs.

Switching between Default and Other Hosts

Most users will usually use the same applications and databases, maybe occasionally using a different database to query data. DBAs should try to match users with default databases, called DEFAULT HOSTS, so they can connect as easily as possible, without needing to specify a network protocol, database name, and so on.

Host is a synonym for the SQL*NET *dbname*. You can switch the current host using the SET INSTANCE command in SQL*DBA.

The *initial default host* is the host identified by a null name, and is the host connected to when SQL*DBA is executed.

The *current default host* is the host you last connected to, either explicitly (via SET INSTANCE) or implicitly (by connecting using no host designation).

Refer to the beginning of Appendix B and the SET INSTANCE command in Appendix B for information on choosing remote instances and performing DBA functions remotely.

Deciding Where to Locate Applications

In a distributed processing environment, the decision of where to place the software application tools is independent of where the data is stored. Important factors in choosing CPUs to place application tools on include:

- what hardware is most convenient for the primary users of the application to access
- level of usage of current hardware choices and the adequacy of response time now and in the future
- availability of network software to connect necessary CPUs
- ease of developing or maintaining application software on a given CPU.

In a distributed database, DBAs have more flexibility in determining where to store data. The issue of data storage affects hardware use, software placement, and data security, insofar as the DBA must guarantee that each user can access the data he requires, but has no more access than he requires.

When designing an optimal distributed database/processing scheme, DBAs should consider at least the following:

- where is the data located currently?
- where will new data (INSERTs) come from primarily?
- how many distinct applications are there?
- from where will application users access the database?

- what hardware is available now or in the future?
- what level of performance is required?
- what are the most important security requirements?

Rather than simply connect previously independent databases and not relocate any data, it is almost always advantageous to move or merge some data. If data is distributed optimally, the benefits include:

- reduced network traffic
- better response time
- potentially less data redundancy
- local control over data used locally
- central control over master copies of data.

As it is usually cheaper to access data stored locally than at a remote location, it is usually desirable to store data where it is used most.

Moving Data with the COPY Command

Data can be transferred using the COPY command in SQL*Plus; see the *SQL*Plus Reference Manual* for the syntax. You can copy data from one remote node to another (as long as they are both part of the distributed database); you need not be directly connected to either database. You can also copy data to a local node from a remote node using:

```
INSERT INTO table_here  
SELECT * FROM table_there@dblink
```

Refer to the *SQL Language Reference Manual* for restrictions that apply to queries in a distributed environment.

Setting Up Database Links

To connect to a database on a remote node, a database link is required. A DBlink contains enough information to uniquely identify a "path" from one database to another. The client database defines and owns the DBLINK and the DBLINK specifies (either explicitly or implicitly) a username and password that are valid in the server database. A DBlink contains the following information:

- network protocol
- nodename
- dbname
- username/password (optional)
- protocol options

as in:

```
D:LONDON:'LONDB':SCOTT/TIGER
```

Note that the path is from no particular UID in the client database but connects directly to a specific UID in the server database. This fact has some implications for the best way to define DBLINKs.

DBLINKs are defined using a SQL statement and are saved in the data dictionary of the client database (see the views USER_DB_LINKS, ALL_DB_LINKS, and DBA_DB_LINKS).

```
CREATE [ PUBLIC ] DATABASE LINK dlname  
[ CONNECT TO username IDENTIFIED BY pwd ]  
USING 'netprefix:dbstring'
```

For example, after the following DBLINK has been defined in a client database

```
CREATE DATABASE LINK LONDB  
CONNECT TO CHARLES IDENTIFIED BY DICKENS  
USING 'D:THAMES'
```

you could enter the following in SQL*Plus:

```
SQL> SELECT * FROM EMP@LONDB  
Paris.
```

You might also create a synonym for the remote table, hiding the existence of the database link, as in:

```
CREATE SYNONYM EMP FOR EMP0@LONDB;  
SELECT * FROM EMP;
```

Private DBLINKs

A private DBLINK is one that is created by one user in a client database for his exclusive use. The data dictionary will only allow the owner to use that particular DBLINK. For another user to access the server database/account, the second user would have to set up his own DBLINK.

Private DBLINKs are generally more secure than public DBLINKs; however, if all users are accessing the same server account, then a public DBLINK to the server account may make more sense.

PUBLIC DBLINKs

A public DBLINK is one that is created in a client database for the use of any user in that database. Thus, all users needing to connect to the specified server UID can use the same DBLINK. A public DBLINK may make more sense if the UID at the server end has general access, as in a "dummy account."

Although this method is somewhat less secure than using private DBLINKs, it is more convenient because it is more general and requires fewer total DBLINKs.

Choosing ORACLE Usernames to Access Remote Data

When you access an ORACLE database, you must connect using a valid ORACLE username/password combination (a UID).

In a distributed database, you need only know your own assigned UID to connect to the local database; all other connections to remote databases are handled by the database for you.

The information required to access a remote database is the same as that required by a database link. Multiple database users may connect remotely using a common username known to all, or each user may use a private connection. The following sections discuss the considerations for both methods.

Using One Central Account for Clients

Many remote clients can connect to a database using one central UID on that database.

Possible advantages of this method include some of convenience:

- Anyone connecting to the database need only remember one UID, knowing that it has proper database privileges.
- Anyone needing to grant database privileges need only grant them to the one UID.

However, the following possible disadvantages affect security:

- The central UID will have privileges on more tables than any one remote user will actually need; thus users will be able to see more data than they require.
- There may be a tendency to over-grant privileges (such as UPDATE, DELETE, INSERT) in order to accommodate all the needs of all remote clients.
- A security breach such as a disclosed password is potentially more dangerous because one UID has so many access privileges to so many database objects.

This technique is implemented by creating a PUBLIC database link containing a specific ORACLE UID, that all users may employ:

```
CREATE DATABASE LINK NODE1  
CONNECT TO NETUSER IDENTIFIED BY NETPASS USING 'dbstring';
```

On the remote host, the DBA would also enter:

```
GRANT CONNECT TO NETUSER IDENTIFIED BY NETPASS
```

and subsequently GRANT appropriate table privileges.

Individual Accounts for Clients

In an alternate method of connecting remote users to a local database, each remote client connects to a database using an individual UID on that database, rather than a standard UID. Two variations exist: each user can have a corresponding UID (a one-to-one association) or several users doing similar tasks can connect using one UID (a many-to-one association). The latter method is a compromise between using one UID for absolutely everyone or using a unique UID for each user.

Possible advantages of this method include:

- By using more UIDs, the grants can suit each UID's individual application requirements, avoiding the tendency to make grants too liberal.
- There is less security exposure, as more UIDs have fewer database privileges and there is no single entry point.

Possible disadvantages of this method include:

- A new UID must be set up or an existing appropriate UID chosen for each remote user requiring access.
- It is somewhat more difficult to administer (more UIDs and DBLINKS to set up and maintain).
- It requires somewhat more coordination between DBAs of the client and server databases.

Each user can have his own database link with a specific embedded ORACLE UID for use on the remote host:

```
CREATE DATABASE LINK MYLINK  
CONNECT TO SCOTT2 IDENTIFIED BY TIGER2 USING 'dbstring';
```

Or, users can reference a PUBLIC database link that does not contain a username and password. In this case the local username and password are used:

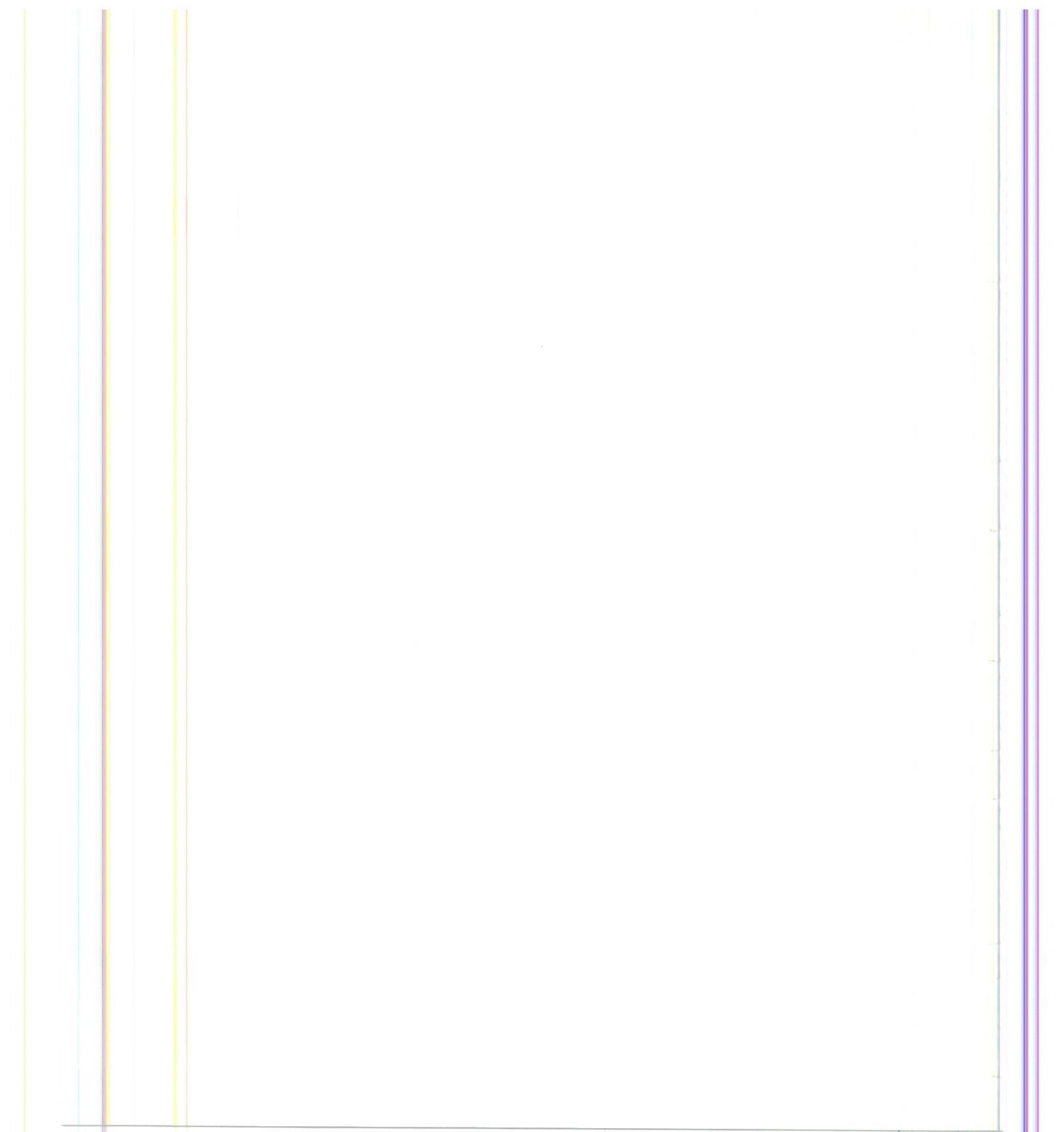
```
CREATE PUBLIC DATABASE LINK HQ USING 'dbstring';
```

See the *SQL Language Reference Manual* for more information.

P A R T

IV

REFERENCE



A

SUMMARY OF CHANGES IN V6.0

This appendix provides more detail about the changes found in the ORACLE RDBMS Version 6. For a shorter list of changes, refer to the Preface.

This list is not intended to be exhaustive nor is it intended to help you upgrade from Version 5 to Version 6. For instructions on upgrading your database, please refer to your *Installation and User's Guide* for Version 6.

Database Terminology

For a summary of terminology differences and equivalences between Version 5 and Version 6, see the end of this appendix. In Version 6 the terms *instance* and *database* have more specific meanings than in Version 5:

instance	the processes and memory required to run and access a database. An instance provides the <i>means of accessing</i> a database.
database	the collection of files, and data within them, such as the data dictionary, that are the <i>data to be accessed</i> .
database system	a database system must have at least one instance and one database.

Rather than *initializing* a database to prepare it for use, you *create a database*, using the SQL statement CREATE DATABASE from SQL*DBA. For more information, see Chapters 1 and 13.

The SQL*DBA Utility

SQL*DBA is a new utility that combines the functionality of the DBA utilities found in Version 5: IOR, ODS, AIJ, CCF. With SQL*DBA you can:

- create a database with a new SQL statement, CREATE DATABASE
- enter any SQL or PL/SQL statement
- start up or shut down one or more instances (local or remote)
- mount (shared or exclusive) and open a database (make the database accessible)
- close and dismount a database (make it unavailable)
- enable or disable archiving of redo data
- backup and restore part or all of the database, while it is running
- take a tablespace offline or bring one online
- monitor ongoing use of the database.

Some SQL*DBA commands are system privileged, which means that in addition to requiring access to the SQL*DBA utility, you also require additional operating system privilege to run these commands. See your *Installation and User's Guide* for details.

DBAs may connect to the database username SYS using the keyword INTERNAL to perform certain database maintenance, such as database creation and altering database files.

Though you may use SQL*DBA to execute any SQL statement including queries, SQL*DBA is not a reporting tool and does not have the formatting abilities of SQL*Plus or any other Oracle reporting product. For more information on SQL*DBA, refer to Chapters 13 and 14, and Appendix B.

SQL*Loader Replaces ODL

In Version 6, SQL*Loader supercedes the ORACLE Data Loader (ODL). SQL*Loader incorporates all the capabilities of ODL and provides new functionality. Refer to the *ORACLE Utilities User's Guide* for more information on SQL*Loader and to your *Installation and User's Guide* for information on upgrading your ODL applications to use SQL*Loader.

Enhanced Data Dictionary

Version 6 introduces a new set of data dictionary views. The views are created by the DBA username SYS.

The new data dictionary views completely supercede and replace the Version 5 views. Views are designed for ease of use and for compatibility with future ANSI/ISO SQL standards. Column names are consistent and mnemonic. The following table lists commonly used V5 views and their new V6 equivalent:

<i>V5 View</i>	<i>V6 View</i>
catalog	all_objects
dtab	dictionary
tab	user_tables and/or user_objects
col	user_tab_columns
columns	all_tab_columns
views	user_views
synonyms	user_synonyms
indexes	user_indexes
clusters	user_clusters

For upward compatibility, the file CATALOG5.SQL is provided to create the V5 data dictionary. If used, this file creates the V5 data dictionary views owned by username SYSTEM. V5 views can coexist with V6 views. Use the file DROPCAT5.SQL to drop the V5 data dictionary.

As Version 6.0 is the last version to support the Version 5 data dictionary, you are advised to modify applications to reference the newer data dictionary rather than the older one.

Database File Structure	The files associated with Version 6 are described below and summarized in the following table.
Control Files	Control files are small mandatory files, automatically created and maintained by the ORACLE RDBMS. Their purpose is to identify and verify database configuration, for example, by identifying and locating the database and redo log files; they do so by recording certain information during database use. One control file must always be read successfully to start a database; multiple identical copies are recommended to ensure maximum database reliability.
Database Files	As with Version 5, a database must consist of at least one database file. In Version 6.0 this file is added to the SYSTEM <i>tablespace</i> , whereas in Version 5 it is added to the SYSTEM <i>partition</i> . In Version 6, the name of this file must be specified in the CREATE DATABASE statement; you can specify multiple database files if you wish.
Redo Log Files	Redo log files are new in Version 6 and have a similar function to the AIJ files in Version 5. Version 6 requires the use of at least two redo log files. You may find that more than two is desirable. Redo log files are also specified in the CREATE DATABASE statement.
INIT.ORA File	Just as in earlier versions, the INIT.ORA file may list several configuration parameters used to limit database access or tune performance. Although the file has the same purpose, most of the parameters have changed or are new. See Appendix D for a complete reference.

The following table compares the structures of a Version 6 database system to those in a Version 5 system:

<i>Version 6</i>	<i>Version 5</i>	<i>Notes</i>
Database file(s)	Database file(s)	essentially identical
Control file(s)	--	new in V6
INIT.ORA file	INIT.ORA files	essentially identical but parameter names change in V6
Redo Log files	--	new and mandatory in V6, similar to AIJ files
--	AIJ files	optional in V5, obsolete in V6
Rollback segments	--	new and mandatory in V6, similar to Before Image file in V5
--	Before Image file	mandatory in V5, obsolete in V6

Database Structures Several database objects have changed, been replaced, or become obsolete.

New Structures The following objects are new.

Tablespaces Similar to Version 5 partitions, tablespaces hold database objects such as tables, clusters, and views. They are the smallest unit of database recovery. The space parameters associated with tablespaces essentially replace the V5 space definitions.

Rollback Segments These structures are used for read consistency, transaction rollback and database recovery. They essentially replace the V5 before image file.

Sequences Created using CREATE SEQUENCE, sequences are used to generate unique keys.

Changed Structures	There are some changes in the structure, storage, or options for the following objects.
Tables	The format of data blocks and rows has changed. New storage parameters and transaction parameters (PCTUSED, INITTRANS and MAXTRANS) can be used when creating database objects.
Views	In Version 6 views require no more space than their entry in the data dictionary. (In Version 5 each view required 3 blocks.)
Indexes	In Version 6 only one index is stored per index segment (in Version 5 all indexes for one table were stored in the same index segment). Also, indexes may be created in a different tablespace than the associated table, allowing for increased performance. All V6 indexes use front (but not rear) compression.
Clusters	Cluster keys no longer need have one NOT NULL column. Cluster indexes must be manually created using the CREATE INDEX command before cluster data can be accessed.
Temporary Segments	Temporary segments are essentially equivalent to Version 5 temporary tables.
NULLs	Sequences of null column values at the end of a row require no storage; however, sequences of null column values followed by a non-null values use 1 byte of storage per null value.
ROWIDs	ROWIDs are identical in format but differ in meaning from Version 5.
Storage	The set of options regarding limits and characteristics of space are called <i>storage parameters</i> . Storage parameters are associated with tablespaces, and with tables, clusters, and indexes. Refer to the <i>SQL Language Reference Manual</i> for a complete description of the STORAGE clause. Some storage parameters carry over from Version 5, and some are new (INITIAL, MINEXTENTS, NEXT, PCTUSED, PCTINCREASE). PCTFREE of 0 is now allowed.

Transaction Control

Several features have been added to give users increased control over their transactions:

- COMMIT and ROLLBACK are now SQL statements (although a setting in SQL*Plus allows them to be treated as in Version 5, as SQL*Plus commands.)
- You can use the new SQL statement SAVEPOINT to name various points within a transaction to which you may rollback.
- Implicit rollbacks result in statement level rollback, rather than transaction level rollback as in ORACLE Version 5.
- You can enable multi-statement read consistency using the SET TRANSACTION READ ONLY statement.

New Lock Modes

In ORACLE Version 6.0 with the transaction processing option, row-level locking is the default mode. See Chapter 12 for an explanation of the following lock modes:

- EXCLUSIVE
- SHARE
- ROW SHARE
- ROW EXCLUSIVE
- SHARE ROW EXCLUSIVE
- SHARE UPDATE.

In ORACLE with the transaction processing option, the only exclusive locks are the row locks acquired when an update takes place. With SELECT FOR UPDATE all rows satisfying the WHERE clause are locked before the first row is fetched. Since the locks are released upon commit you cannot fetch from a SELECT FOR UPDATE cursor after a commit.

ORACLE V6 without the transaction processing option has the same locks, except an exclusive table lock is acquired when an update occurs. In Version 5, row locks were placed when each row was fetched.

Password Encryption

Because V6 uses a new password encryption routine, the DBA must reset all user passwords after performing a database import.

Error Codes

Many error messages are new and some have become obsolete. If your programs have any hardcoded checks for specific error numbers, please check to verify that the error has not been changed.

Changes in SQL

The Oracle implementation of the SQL language has several changes in V6. Refer to the *SQL Language Reference Manual* for a complete description of any SQL statement.

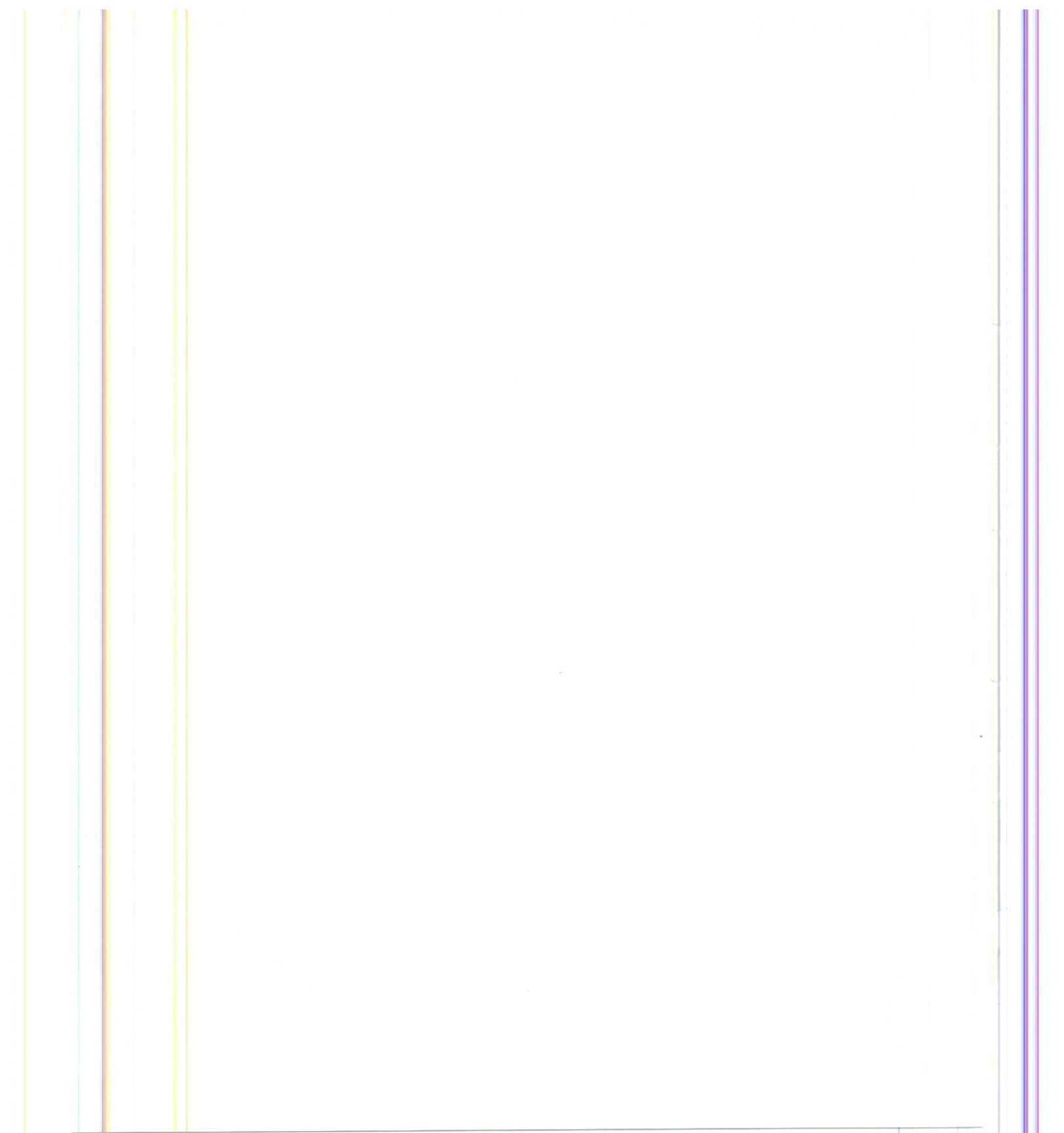
- The list of reserved words has changed; see the *SQL Language Reference Manual*.
- There are several new SQL commands, including:

```
CREATE | DROP | ALTER DATABASE
CREATE | DROP | ALTER ROLLBACK SEGMENT
CREATE | DROP | ALTER SEQUENCE
CREATE | DROP | ALTER TABLESPACE
SAVEPOINT
SET TRANSACTION READ ONLY
```
- Several commands have been modified, in particular, those relating to storage specifications and lock modes.
- Due to changes in database structure, all commands referencing partitions or space definitions are obsolete.
- The new CREATE SEQUENCE statement automatically generate a series of unique ascending or descending integers. The pseudo-columns CURRVAL and NEXTVAL are used to fetch newly generated key values.
- Referential integrity syntax is supported but not enforced. That is, additional information is stored in the data dictionary but is not yet used for data validation. New CREATE TABLE clauses include:
 - DEFAULT
 - NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY ... REFERENCES
 - CHECK
- Two new National Language Support functions are supported (NLSSORT and REPLACE).
- Execution errors cause statement level rollback rather than transaction level rollback as in Version 5.
- Nulls sort highest on ascending sequences and lowest on descending sequences.

Version 5 and Version 6 Terminology

The following table contains "loose" equivalents for features of Version 5.1 which have changed form, name, or means of access.

<i>Version 5.1 Feature</i>	<i>Version 6.0 Equivalent</i>
partitions	obsolete, replaced by tablespaces
space definitions	obsolete, replaced by storage clause in CREATE, ALTER
Before Image file	obsolete, replaced by rollback segments
ODS utility	SQL*DBA command: MONITOR
System Global Area (SGA)	identical in Version 6.0
4 background processes (ARH, CLN, BIW, BWR)	up to 5 background processes (SMON, PMON, DBWR, LGWR, and optional ARCH)
optional use of AIJ	obsolete, replaced by mandatory use of redo log files, used with optional archiving of redo data.
(none)	ARCHIVELOG mode, NOARCHIVELOG mode
Recovering AIJ files	SQL*DBA command: RECOVER
CCF utility	SQL statement CREATE DATABASE
IOR INIT	SQL statement CREATE DATABASE
IOR WARM	SQL*DBA command STARTUP
(none)	mounting/dismounting a database
(none)	opening/closing a database
IOR SHUT	SQL*DBA command SHUTDOWN
INIT.ORA file	essentially identical in V6.0
cluster indexes	must be manually created in V6.0
data/index segments	essentially the same, but 1 segment/index
temporary tables	essentially the same but called segments
(none)	control file (associated with a database)
DTAB data dictionary table	DICTIONARY view



B

THE SQL*DBA REFERENCE

SQL*DBA is a tool for helping database administrators manage and monitor a database. SQL*DBA can be used to perform the following tasks:

- start and stop an ORACLE instance
- mount, dismount, open, and close an ORACLE database
- monitor realtime use and performance of an ORACLE database
- perform backup and recovery of database logs and data
- execute any SQL statement.

This appendix is the primary reference for the use of SQL*DBA and contains full information about using SQL*DBA, entering SQL*DBA commands, and an alphabetical reference of all SQL*DBA commands.

Invoking SQL*DBA

To invoke SQL*DBA enter the command SQLDBA at the operating system prompt. SQL*DBA responds with a prompt:

```
you enter:          sqldba  
you will see:      SQLDBA>
```

You may optionally indicate what type of terminal you are using, as in:

```
SQLDBA TERMINAL=VT100
```

where *terminal type* is a standard abbreviation recognized by your operating system for a terminal, such as VT100 or IBM3270. Recognized terminal types correspond to the terminal names used by the CRT utility. For more information on terminal types and where SQL*DBA expects to find the .CRT files, refer to your *Installation and User's Guide*. For more information on the CRT utility, refer to the *ORACLE Utilities User's Guide*.

You may use SQL*DBA interactively or you may enter a SQLDBA command line at the operating system level. Variations of using SQL*DBA are described in the following sections.

When using SQL*DBA interactively, you can enter four types of responses to the SQLDBA prompt:

- a SQL*DBA command, with optional arguments
- a SQL statement
- a PL/SQL block
- a command to run a command file.

Most SQL*DBA commands require you to first connect to a database with an ORACLE username and password. An obvious exception is the CONNECT command.

If SQL*DBA detects an error in your entry, it displays the entire line with an indication of the location of the error.

Entering SQL*DBA Commands

Several SQL*DBA commands are available. SQL*DBA commands:

- may be longer than one line if a back slash is used as the last non-blank on a line to indicate continuation
- require no punctuation or terminators.

Individual syntax descriptions for each SQL*DBA command appear at the end of this appendix.

SQL*DBA commands which are not system privileged can be executed by any person having access to the SQL*DBA program.

System Privileged SQL*DBA Commands

Some SQL*DBA commands are *system privileged* because they can have powerful effects on the database. For example, the commands STARTUP, SHUTDOWN, and CONNECT INTERNAL are all system privileged.

These commands are limited to users who have been given *both*:

- access to use SQL*DBA, and
- *operating system access* to use the privileged SQL*DBA commands.

These users should be limited in number and are assumed to be DBAs. Because the details about both types of access are operating system specific, refer to the *Installation and User's Guides* for additional information.

Entering SQL Commands

You can execute any SQL statement from within SQL*DBA if you are connected to a database. A SQL statement ends and may be executed if either of the following is true:

- a slash is the first and only character on an input line
- a semicolon is the last character on an input line.

Thus, the following are valid SQL*DBA entries:

```
SQLDBA> CONNECT scott/tiger  
  
SQLDBA> SELECT * FROM DICTIONARY  
2> WHERE TABLE_NAME LIKE 'USER_%';  
  
SQLDBA> SELECT * FROM SCOTT.EMP  
2> /
```

Ordinary SQL security applies to any SQL statement; that is, the username connected to the database must have the appropriate privilege to execute that statement. For example, if usernamer SCOTT is using SQL*DBA and does not have DBA privilege, then he must have SELECT access on NANCY's tables in order to query table NANCY.PAYROLL.

Note that unlike SQL*Plus, SQL*DBA does not store SQL statements in a buffer. Thus, SQL*Plus buffer commands do not work in SQL*DBA.

Entering PL/SQL Commands

If you have installed the ORACLE RDBMS with the tps option, you may also enter any PL/SQL statements in response to the SQL*DBA prompt. Use a period(.) to terminate entry of a PL/SQL or SQL statement, in order to end entry without execution. Do not terminate lines beginning with DECLARE or BEGIN with a semicolon.

To execute a PL/SQL block, enter a slash on a line by itself.

Running Command Files

To run a command file, enter the filename after an at-sign, as in:

```
SQLDBA> @beginday
```

If not specified, the file's extension or filetype is assumed to be SQL. Do not follow the filename with a slash or a semicolon. Within the command file you should terminate all SQL statements with a semicolon or slash if you wish them to be executed (because SQL*DBA has no buffers like SQL*Plus, commands are not stored in a buffer). Command files may call other command files; the depth to which command files may be nested is operating system specific.

Entering Commands from the Operating System

You can run SQL*DBA commands without entering the SQL*DBA environment, by following the SQLDBA command with a command string. The command string may be a SQL*DBA command or the name of a file to run. The syntax is:

```
SQLDBA [ COMMAND="command text" ] [ TERMINAL=terminal type]
```

For example:

```
SQLDBA command=""startup dba pfile=myinit.ora""
```

(Some operating systems, such as VMS, require extra quotations, as shown above, in order to include quotations in the string.)

Using SQL*DBA in a Distributed Environment

If SQL*Net is also installed, SQL*DBA can be used to connect to other databases on nodes of the communications network. Thus, you can use SQL*DBA to start an instance on a remote node.

Use SET INSTANCE to specify the database specification of the instance with which you want to work. This instance will be used for subsequent STARTUP, SHUTDOWN, MONITOR, CONNECT commands and all SQL statements which do not use explicit database specifications.

See Chapter 22 and the SET INSTANCE and SHOW INSTANCE commands in this chapter for more details.

Monitoring Database Use with MONITOR

Using the MONITOR command you can view ongoing use of the database. For example, you can see:

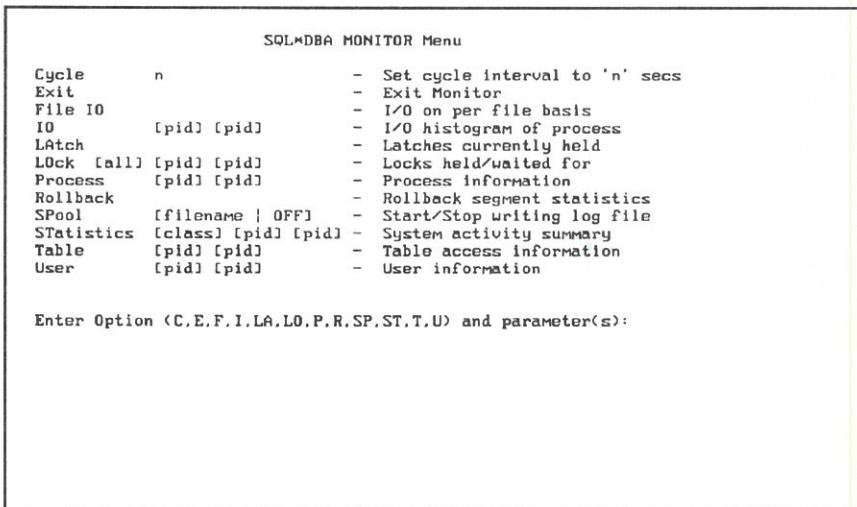
- ORACLE usernames of current database users
- which programs they are running
- which users are waiting for other users
- which tables they are accessing
- a summary of I/O
- cumulative statistics about database usage.

You can invoke particular monitor screens directly, as in:

```
SQLDBA> MONITOR LOCK
```

or you can use the MONITOR menu by simply entering MONITOR. The menu appears in Figure B-1.

FIGURE B-1
The MONITOR Menu



As MONITOR provides a continuously updated display of database activity, you can monitor activity over time to spot patterns of normal or abnormal database use.

MONITOR displays are updated at intervals. The default interval is every 5 seconds; you can change the interval using the SQL*DBA command SET CYCLE. Although valid intervals may range from 1 to

3600 seconds, because some overhead is associated with refreshing the screen, a minimum interval of 5 seconds is recommended.

You can send output to a file by using the SQL*DBA command SPOOL. The various MONITOR displays are described in the following sections.

Dynamic Performance Tables used by MONITOR

In addition to using the MONITOR displays you can directly access the data shown in the displays. The data is stored in pseudo-tables, called *dynamic performance tables*. Although these tables are stored only in memory (they are not physically on disk), they can be queried in SQL SELECT statements similar to read-only tables.

These tables are owned by SYS and their names are prefixed with V_\$. Views are created upon them begin with V\$'s; it is these views that users should access. For a listing of these views, refer to Appendix E.

Granting Other Users Access to MONITOR Screens

By default, only user SYS has access to the dynamic performance tables that provide the information shown in the MONITOR displays. To grant access to all other users, SYS can run the file MONITOR.SQL. This file contains the necessary grants to allow all users to view the MONITOR screens. (They must also have access to SQL*DBA in order to use the SQL*DBA command MONITOR.)

You may modify MONITOR.SQL if you wish to grant access to these tables to a select group of users rather than to PUBLIC (all users).

Characteristics of All Displays

The sample displays in this chapter come from a VAX/VMS system; displays may vary by operating system.

The date and time of the display appear in the upper right hand corner. If you have used SET INSTANCE, the current node and instance identifiers appear in the upper left hand corner.

By default the screens will cycle through individual displays for each process. You can see a display for a single process or a range of processes by specifying an individual process identifier number (PID) or a range of numbers. You can also request statistics on a system-wide basis (showing statistics for all processes), by specifying PID 0.

Process 1 is used during startup and not thereafter. Processes 2 through 5 are always present and represent the four background processes (DBWR, LGWR, SMON, PMON). Process 6 represents ARCH, if it is enabled. These processes always appear in the same order.

Many object IDs are represented in hexadecimal so that they require fewer characters on the screen.

If you have trouble getting the MONITOR displays to display properly on your screen, you should check the following:

- Do you have proper access to SQL*DBA?
- Do you have access to the underlying dynamic performance tables?
- Are you running SQL*DBA using the proper terminal (CRT) definition?

The MONITOR PROCESS Display

A sample MONITOR PROCESS display appears in Figure B-2. This screen summarizes information about all processes, including the background processes, currently accessing the database through the specified instance.

Use this screen to determine the operating system IDs that are accessing a database from this instance, and what ORACLE programs they are executing. To see what ORACLE username a process is currently using and the last SQL statement executed, see the display for MONITOR USER.

FIGURE B-2
Sample MONITOR PROCESS Display

ORACLE Process Monitor						Tue Nov 1 16:45:33
ORACLE PID	System PID	System User Name	Terminal	Program		
2 ZC000B92	MMOORE	UNKNOWN	\$2\$DUAZZ:[U6KNLDUE, SHELLY, ENU]SRV.EXE:Z			
3 ZC000793	MMOORE	UNKNOWN	\$2\$DUAZZ:[U6KNLDUE, SHELLY, ENU]SRV.EXE:Z			
4 ZC000B95	MMOORE	UNKNOWN	\$2\$DUAZZ:[U6KNLDUE, SHELLY, ENU]SRV.EXE:Z			
5 ZC000796	MMOORE	UNKNOWN	\$2\$DUAZZ:[U6KNLDUE, SHELLY, ENU]SRV.EXE:Z			
6 ZC001132	SDIMICK2		\$1\$DUAZ9:[SDIMICK.TP1]TP1PROC.EXE			
7 ZC001550	SDIMICK2		\$1\$DUAZ9:[SDIMICK.TP1]TP1PROC.EXE			
8 ZC001451	SDIMICK2		\$1\$DUAZ9:[SDIMICK.TP1]TP1PROC.EXE			
9 ZC001940	SDIMICK2		\$1\$DUAZ9:[SDIMICK.TP1]TP1PROC.EXE			
10 ZC001653	SDIMICK2		\$1\$DUAZ9:[SDIMICK.TP1]TP1PROC.EXE			
11 ZC001854	SDIMICK2		\$1\$DUAZ9:[SDIMICK.TP1]TP1PROC.EXE			
12 ZC001CS2	SDIMICK2		\$1\$DUAZ9:[SDIMICK.TP1]TP1PROC.EXE			
13 ZC001C55	SDIMICK2		\$1\$DUAZ9:[SDIMICK.TP1]TP1PROC.EXE			
15 ZC00136D	SDIMICK2		RTA14: \$2\$DUAZZ:[U6KNLDUE, SHELLY, ENU]SQLDBA.EXE:Z			

Note that the first four processes (beginning with Process 2) are always the four background processes (DBWR, LGWR, SMON, PMON) and if ARCH is enabled, it is the fifth shown. These processes all run the same program. The username shown in the third column is not the ORACLE username, but the name of the operating system account or ID from which that user is accessing the database.

The MONITOR USER Display A sample MONITOR USER display appears in Figure B-3.

FIGURE B-3
Sample MONITOR USER
Display

```
ORACLE User Monitor      Tue Nov 1 16:48:12
$1$DUAZ9:[SDIMMICK.TP1JTP1PROC.EXE

Process Information
ORACLE PID: 6          Latch Wait: NONE
System PID: 2C001132    Lock Wait: NONE
User Name: SDIMMICKZ
Terminal:
Status: FOREGROUND

Session Information
Session #: 1
User Name: TP1
User ID: 5
Command: UPDATE
```

The upper half of the display indicates the process number currently assigned to the username, the system ID and account or username, and the system status. This information is similar to that shown in the MONITOR PROCESS display.

The lower half of the screen displays ORACLE information, including username, session number (currently always 1), and most recent SQL statement executed.

The MONITOR TABLE Display

A sample MONITOR TABLE display appears in Figure B-4. This display shows all tables currently accessed by currently parsed cursors. The process identified by column 1 is accessing the table named in the last column; to find out whether that PID is the owner of the table or not, you must refer to the MONITOR USER display to see which ORACLE username corresponds to the PID.

The example is a system-wide display; you may also request the display on a per user basis.

FIGURE B-4
Sample MONITOR TABLE Display

PID	Obj#	Owner ID	Owner Name	Table Name
13	14f	5	TP1	BRANCH
13	14e	5	TP1	TELLER
11	14f	5	TP1	BRANCH
11	15b	5	TP1	HISTORY
12	14d	5	TP1	ACCOUNT
12	14e	5	TP1	TELLER
12	15b	5	TP1	HISTORY
12	14f	5	TP1	BRANCH
13	14d	5	TP1	ACCOUNT
13	15b	5	TP1	HISTORY
10	14f	5	TP1	BRANCH
10	14e	5	TP1	TELLER
10	15b	5	TP1	HISTORY
10	14d	5	TP1	ACCOUNT

The number in the second column is the hexadecimal representation of the object's database ID. This hexadecimal ID also appears in the MONITOR LOCKS display. If converted to decimal this number is the object's ID as found in the database.

The MONITOR LOCK Display

Three sample MONITOR LOCK displays appear in Figures B-5 through B-7. Refer to Chapter 12 for a discussion of the types of locks and the meaning of share and exclusive locks.

FIGURE B-5
Sample MONITOR LOCKS Display

ORACLE Lock Monitor		Tue Nov 1 16:54:27										
		UPPER case = owner					Lower case = waiter					
18 Resources 13 Processes		S=share X=exclusive		L=rou-S			R=row-X		C=S/row-X			
Resource	Object ID	Owner	Share	Exclusive	Row	Share	Row	Share	Exclusive	Row	Share	Exclusive
RT.....1.....	0		X									
RW.10002c1.....	0						X					
RW.100030f.....	d					X						
RW.100030f.....	f						C					
RW.1000311.....	d							X				
RW.100039f.....	0					X						
TD.....14d.....	0		S	S	S	S	S	S	S	S	S	S
TD.....14e.....	0		S	S	S	S	S	S	S	S	S	S
TD.....14f.....	0		S	S	S	S	S	S	S	S	S	S
TD.....15a.....	0		S	S	S	S	S	S	S	S	S	S
TM.....14d.....	0		R	R	R	R	R	R	R	R	R	R
TM.....14e.....	0		R									
TM.....14f.....	0		R									
TM.....15a.....	0		R	R			R	R				
TX.....11.....	57							X				
TX.....16.....	56								X			
TX.....25.....	56									X		
TX.....28.....	56										X	

A summary of some of the abbreviations follows:

- | | |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RT locks | Redo log buffer lock (held by LGWR). |
| TD locks | TD locks are DDL locks and may be held in S or X mode. The second column after the lock name indicates the object ID in hexadecimal. This number also appears in the TABLES display along with the object's name. |
| TM locks | TM locks are data locks on tables. The second column after the lock name indicates the object ID in hexadecimal. This number also appears in the TABLES display along with the object's name. |
| TS locks | TS locks are locks on temporary segments. The second column after the lock name indicates the identifier of the temporary segment. |
| TX locks | TX locks are transaction locks; the presence of a TX lock indicates a process holding a row lock. |

FIGURE B-6
Sample MONITOR LOCKS
Display

		ORACLE Lock Monitor													Tue Nov 1 16:58:54	
		UPPER case = owner lower case = waiter														
		S=share X=exclusive L=row-S R=row-X C=S/row-X														
<hr/>																
CF.....0.....0		X														
CS.....0.....0		S														
RT.....1.....0		X														
RW.10002b6.....0																X
RW.10002c1.....0																X
RW.10002ca.....0																X
RW.10002ed.....0																X
RW.10002f0.....0																X
RW.100030d.....9																C
TD....14d.....0		S	S	S	S	S	S	S	S	S	S	S	S	S	S	
TD....14e.....0		S	S	S	S	S	S	S	S	S	S	S	S	S	S	
TD....14f.....0		S	S	S	S	S	S	S	S	S	S	S	S	S	S	
TD....15a.....0		S	S	S	S	S	S	S	S	S	S	S	S	S	S	
TM....14d.....0		R	R	R	R	R	R	R	R	R	R	R	R	R	R	
TM....14e.....0		R	R	R	R	R	R	R	R	R	R	R	R	R	R	
TM....15a.....0		R	R	R	R	R	R	R	R	R	R	R	R	R	R	
TX....1e.....63		R	R	R	R	R	R	R	R	R	R	R	R	R	R	
TX....1f.....63		X														
TX....21.....62		X														
TX....23.....63		X														

FIGURE B-7
Sample MONITOR LOCKS
Display

		ORACLE Lock Monitor													Tue Nov 1 16:59:32	
		UPPER case = owner lower case = waiter														
		S=share X=exclusive L=row-S R=row-X C=S/row-X														
<hr/>																
RT.....1.....0		X														
RW.10002b4.....0																X
RW.10002c8.....0																X
RW.10002de.....0																X
RW.10002eb.....0																X
RW.10002ee.....0																X
RW.1000310.....C																C
TD....14d.....0		S	S	S	S	S	S	S	S	S	S	S	S	S	S	
TD....14e.....0		S	S	S	S	S	S	S	S	S	S	S	S	S	S	
TD....14f.....0		S	S	S	S	S	S	S	S	S	S	S	S	S	S	
TD....15a.....0		S	S	S	S	S	S	S	S	S	S	S	S	S	S	
TM....14d.....0		R	R	R	R	R	R	R	R	R	R	R	R	R	R	
TM....14e.....0		R	R	R	R	R	R	R	R	R	R	R	R	R	R	
TM....15a.....0		R	R	R	R	R	R	R	R	R	R	R	R	R	R	
TX....1b.....65		R	R	R	R	R	R	R	R	R	R	R	R	R	R	
TX....1f.....65		X														
TX....10.....65		X														
TX....21.....64		X														
TX....23.....65		X														
TX....3c.....65		X														

The MONITOR LATCH Display

MONITOR LATCH can be used to see how internal data structure latches are being used. *Latches* protect shared data structures in the SGA. A user or background process acquires a latch when working with a structure and releases the latch when finished with the structure. Each latch protects a different set of data, identified by the name of the latch.

A sample MONITOR LATCH display is shown in Figure B-8.

FIGURE B-8
Sample MONITOR LATCH Display

Latch Name	Holder PID	ORACLE Latch Monitor			Tue Nov 1 17:04:04	
		Total	Willing-to-Wait-Requests	No-Wait-Requests	Total	Successes
process allocation	399	1	398	0	0	0
session allocation	259341	14	259338	0	0	0
messages	188804	2692	188534	0	0	0
enqueues	299178	3650	295973	0	0	0
cache buffers chains	739510	4972	735253	55002	54661	
cache buffers lru ch	68235	5976	67748	177362	176054	
cache buffer handles	71600	141	71462	0	0	
multiblock read obje	2144	0	2144	0	0	
inter-instance buffe	0	0	0	0	0	
system commit number	181663	168	181502	0	0	
Archive control	0	0	0	0	0	
Redo allocation	122233	449	121828	0	0	
Redo copy	0	0	0	6849	6849	
DML/DDL allocation	79360	153	79228	0	0	
transaction allocati	16075	15	16060	0	0	
undo global data	14627	4	14623	0	0	
sequence cache latch	585	2	584	0	0	
row cache objects	258989	141	258937	0	0	

The values of most interest in this display are those in the *timeouts* column. These values signify the ratio of timeouts per wait. The implementation of latching and the timeout algorithm is very operating system dependent; on some operating systems, processes timeout without waiting for a resource at all, while on other systems, processes will make several attempts to obtain a latch before timing out. If the value for timeouts is high in the latter type of system, it is more indicative of a problem than in the former type of system.

The values shown in the *immediate* column indicate latches that were obtained immediately.

The numbers shown in the columns *immediate* and *timeouts* need not necessarily add to the *total* column, because a single request can timeout multiple times and processes can terminate while obtaining a latch.

In particular, you will be interested in the values for the following latches:

- enqueue
- cache buffers chains
- cache buffers lru chain
- cache buffer handles
- redo allocation
- redo copy

The other values are primarily for use by Oracle support personnel.

If the values for the *cache...* parameters are high relative to your operating system and database characteristics then you can reduce the INIT.ORA parameters DB_BLOCK_MAX_CLEAN_PCT and DB_BLOCK_TIMEOUT_WRITE_PCT.

The MONITOR STATISTICS Display

Use the MONITOR STATISTICS command to see runtime statistics about system use and performance. By default, statistics are shown for all processes (system-wide); you can also request the screens for an individual process or range of processes. All available statistics can be displayed using the command MONITOR STATISTICS ALL, or groups of statistics can be requested, using the following syntax:

```
MONITOR STATS [ USER | ENQUEUE | CACHE | REDO | ALL ]
```

With some exceptions, as noted, all statistics are shown as rates per second. The following five columns appear:

CUR	The current value for this statistic and this process (or all processes). This value is the value as of the last cycle's update.
AVG	This value represents the average value for this statistic since the screen was displayed. If the screen is on a user process basis, the average is shown for that process; if the screen represents all processes, then the average is across all processes.
MAX	This value represents the highest value in the cycle for a given statistic since the screen was displayed.
MIN	This value represents the lowest value in this cycle for a given statistic since the screen was displayed.
TOT	This number represents the total value for each statistic since the screen was displayed, on either a process or global basis.

Figure B-9 shows the display resulting from MONITOR STATS USER.

FIGURE B-9
Sample MONITOR STATS User Display

Statistic Name	ORACLE Statistics Monitor					Tue Nov 1 16:53:25		
	ORACLE PID: 0	Session #: 0	User Name: SYSTEM STATISTICS	CUR	AVG	MAX	MIN	TOT
logons	0	null	null	null		69		
current logons	0	null	null	null		13		
cumulative opened cu	0	null	null	null		11841		
current open cursors	0	null	null	null		33		
user commits	0.00	2.68	5.00	0.00		287		
user rollbacks	0.00	0.00	0.00	0.00		0		
user calls	0.40	13.79	24.00	0.40		1475		
recursive calls	0.00	1.50	18.60	0.00		161		
messages sent	0.20	1.79	3.40	0.20		191		
messages received	0.20	1.79	3.40	0.20		191		
background timeouts	0.20	0.07	0.20	0.00		7		

background timeouts	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
cumulative opened cursors	The number of all cursors opened since warmstart (including cursors that have since closed). This number is not reset to zero when the display is called, but is cumulative from the last warmstart.
current logons	This row is more useful on a system-wide basis. The number in the CUR column is the number of new logons this cycle (if shown on a per-process basis, always 1 per process). It always equals the corresponding number in <i>logons</i> .
	The number in the TOT column is the number of current logons (as of the start of the cycle).
current open cursors	The number of currently open cursors. This row is more useful on a system-wide basis.
logons	On a system-wide basis, the number in the CUR column is the number of new logons this cycle and always equals the corresponding number in <i>current logons</i> . On a per-process basis this number is always 0 or 1.
	The number in the TOT column is the number of processes that have ever logged on (including those that have since logged off). It is not reset to zero when the display is called, but is cumulative from the last warmstart.
messages received	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
messages sent	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
recursive calls	The number of recursive calls including data dictionary operations. A high number indicates that information is not being found often enough in the dictionary cache. This could be because parsing is occurring (too) often or because the cache is not large enough. You should consider increasing the INIT.ORA parameters beginning with DC.
user calls	The number of user calls processed.
user commits	The number of transactions for this process or system-wide. This number should equal <i>user calls</i> minus <i>user rollbacks</i> .

user rollbacks The number of rollbacks for this process.

Figure B-10 shows the display for MONITOR STATS ENQUEUE.

FIGURE B-10
Sample MONITOR STATS
ENQUEUE Display

Statistic Name	ORACLE Statistics Monitor					Tue Nov 1 17:34:07			
	ORACLE PID: 0	Session #: 0	User Name: SYSTEM	STATISTICS	TOT	CUR	AVG	MAX	MIN
enqueue timeouts	0.00	0.20	0.60	0.00	3				
enqueue waits	0.00	0.47	1.20	0.00	7				
enqueue deadlocks	0.00	0.00	0.00	0.00	0				
enqueue requests	25.00	22.47	26.20	16.20	337				
enqueue conversions	0.40	1.33	3.20	0.40	20				
enqueue releases	25.20	21.47	26.00	13.20	322				

- | | |
|---------------------|-----------------------------------------------------------------------------------------------|
| enqueue conversions | You may disregard this statistic, as it is primarily for the use of Oracle Support personnel. |
| enqueue deadlocks | The number of process deadlocks that occurred due to enqueues (locks) for DDL operations. |
| enqueue releases | You may disregard this statistic, as it is primarily for the use of Oracle Support personnel. |
| enqueue requests | The number of times enqueues (locks) were requested. |
| enqueue timeouts | The number of times an enqueue (lock) was not granted immediately. |
| enqueue waits | You may disregard this statistic, as it is primarily for the use of Oracle Support personnel. |

Figures B-11 and B-12 show the two screens for MONITOR STATISTICS CACHE.

FIGURE B-11
Sample MONITOR STATS CACHE Display (Page 1)

Statistic Name	ORACLE Statistics Monitor					Tue Nov 1 16:41:14
	ORACLE PID: 0	Session #: 0	User Name: SYSTEM	STATISTICS	TOT	
	CUR	AUG	MAX	MIN		
db block gets	40.70	28.46	44.18	0.00	3969	
consistent gets	25.90	12.74	28.40	0.00	2472	
physical reads	6.60	3.60	7.50	0.00	699	
physical writes	9.90	4.39	9.90	0.00	852	
db block changes	42.90	21.35	45.10	0.00	4141	
change write time	0.00	0.00	0.00	0.00	0	
consistent changes	2.40	1.05	2.60	0.00	204	
write complete waits	0.90	0.38	1.45	0.00	73	
write wait time	0.00	0.00	0.00	0.00	0	
buffer busy waits	0.00	0.00	0.00	0.00	0	
busy wait time	0.00	0.00	0.00	0.00	0	
redo synch writes	3.10	1.53	3.50	0.00	296	
redo synch time	0.00	0.00	0.00	0.00	0	
DBWR exchange waits	0.00	0.00	0.00	0.00	0	
exchange deadlocks	0.00	0.00	0.00	0.00	0	
free buffer requests	15.30	7.85	17.80	0.00	1523	
free buffer scans	15.30	7.85	17.80	0.00	1523	
free buffer inspects	60.00	29.74	71.20	0.00	5769	
free buffer waits	5.50	2.67	6.60	0.00	518	
free wait time	0.00	0.00	0.00	0.00	0	

- busy wait time You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
- buffer busy waits The number of times that a user process wanted a buffer in a mode which is incompatible with the current use of that buffer. A high value indicates block level contention.
- change write time You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
- consistent changes The number of log changes to produce read consistent blocks.
- consistent gets The number of logical reads in read consistent mode. The sum of this number and *db block gets* equals logical reads.
- db block changes The number of logical changes to current database blocks.
- db block gets The number of requests for the current copy of a block. This number plus *consistent gets* equals logical reads.

dbwr exchange waits	Used multi-instance only, the number of times the DBWR was requested to exchange blocks between instances. Useful only when shown for the DBWR process.
exchange deadlocks	The number of times that a process detected a potential deadlock when exchanging two buffers and so raised an internal, restartable error. Index operations are the only operation which performs exchanges. If this number is high, increase the parameters GC_DB_LOCKS and DB_BLOCK_BUFFERS.
free buffer inspected	The number of buffers skipped in the buffer cache in order to find a free buffer. High values potentially indicate that user processes are doing too much work and DBWR not enough. To reduce high values, increase the INIT.ORA parameter DB_BLOCK_MAX_CLEAN_PCT or decrease the parameter DB_BLOCK_MAX_SCAN_PCT.
free buffer requested	The total number of free buffers needed.
free buffer scans	The total number of times the LRU list was scanned to find a free buffer.
free buffer waits	The number of times a process needed a free buffer (in order to read from the disk or to make a read consistent block) and a free buffer was not available.
free wait time	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
physical reads	The number of actual reads of database blocks from disk.
physical writes	The number of actual writes to disk made by DBWR. For multiple-user systems is always 0, except for DBWR.
redo synch time	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
redo synch writes	The number of times the redo is forced to disk, usually for transaction commit.
write complete waits	The number of times a process waited for DBWR to write a current block before making a change.
write wait time	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.

FIGURE B-12
Sample MONITOR STATS
CACHE Display (Page 2)

Statistic Name	CUR	AVG	MAX	MIN	TOT
<i>...continued (page 2)</i>					
dbwr timeouts	0.00	0.16	0.40	0.00	39
dbwr free needed	0.50	0.40	1.00	0.00	99
dbwr free low	0.00	0.02	0.30	0.00	5
dbwr buffers scanned	5.00	5.20	10.91	1.20	1274
dbwr checkpoints	0.40	0.08	0.40	0.00	19
consistent forceouts	0.00	0.00	0.00	0.00	0

- | | |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| consistent forceouts | You may disregard this statistic, as it is primarily for the use of Oracle Support personnel. |
| dbwr buffers scanned | The number of buffers in the buffer cache looked at by DBWR. |
| dbwr checkpoints | The number of checkpoints completed by DBWR. |
| dbwr free low | The number of times DBWR was notified because a user process checked DB_BLOCK_MAX_MOD_PCT buffers while searching for a free buffer. |
| dbwr free needed | The number of times DBWR was notified because no free buffers were located by a user process. |
| dbwr timeouts | The number of timeout wakeups of DBWR. |

Figure B-13 shows the screen for MONITOR STATS REDO.

FIGURE B-13
Sample MONITOR STATS
REDO Display

Statistic Name	ORACLE Statistics Monitor			Tue Nov 1 17:29:54	
	Session #: 0	User Name:	SYSTEM STATISTICS	MIN	TOT
	CUR	AVG	MAX		
redo entries	4.60	16.12	37.40	1.00	661
redo size	836.88	2837.66	6515.00	362.00	116344
redo entries linearized	0.00	0.00	0.00	0.00	0
redo buffer allocations	0.20	0.34	0.60	0.00	14
redo small copies	4.20	15.88	37.00	1.00	651
redo wastage	99.40	401.98	791.00	0.00	16481
redo writer latching	0.00	0.00	0.00	0.00	0
redo writes	1.00	2.12	4.20	0.00	87
redo blocks written	3.40	6.51	15.40	0.00	267
redo write time	0.00	0.00	0.00	0.00	0
redo log switch wait	0.00	0.00	0.00	0.00	0
redo chunk allocations	0.00	0.07	0.20	0.00	3
redo log space requests	1.00	1.34	3.40	0.00	55
redo log space wait	0.00	0.00	0.00	0.00	0
redo log switch intervals	0.00	0.00	0.00	0.00	0

redo blocks written	The total number of redo blocks written.
redo buffer allocation retries	The total number of retries necessary to allocate space in the redo buffer. Retries are needed because either the redo writer has fallen behind or because an event, such as a log switch, is occurring.
redo chunk allocations	The number of times the redo writer allocated another section of the redo file to this instance.
redo entries	The number of redo entries generated
redo entries linearized	The number of entries less than or equal to REDO_ENTRY_PREBUILD_THRESHOLD. Building these entries requires additional CPU but increases concurrency on multiple-user systems.
redo log space requests	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
redo log space wait	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
redo log switch interrupts	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
redo log switch wait	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.

redo size	The amount in bytes of redo generated.
redo small copies	The total number of entries less than or equal to LOG_SMALL_ENTRY_MAX_SIZE. These entries are copied under the protection of the allocation latch, eliminating the necessity of the copy latch. This statistic is generally only interesting for a multi-processor.
redo wastage	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
redo writer latching time	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.
redo writes	The total number of redo writes.
redo write time	You may disregard this statistic, as it is primarily for the use of Oracle Support personnel.

The MONITOR I/O Display

A sample MONITOR IO display appears in Figure B-14. This display indicates all currently connected processes (including the background processes) and the percent of I/O that each process is responsible for:

- in the last interval (on the left, indicated by Interval)
- since the last warmstart (on the right, indicated by Cumulative)

Note that the cumulative histogram's results may not add up to 100, because processes that were previously connected are also responsible for cumulative I/O.

FIGURE B-14
Sample MONITOR I/O Display

ORACLE System I/O Distribution Monitor Tue Nov 1 17:05:48									
Interval					Cumulative				
% Logical Reads	% Physical Reads	% Logical Writes	% Logical Reads	% Physical Reads	% Logical Writes	% Logical Reads	% Physical Reads	% Logical Writes	% Logical Reads
0	100	0	0	100	0	0	100	0	100
-	-	-	2(1)	=	=	=	=	=	=
=	==	=	3(1)	=	=	=	=	=	=
=	=	=	4(1)	=	=	=	=	=	=
-	==	-	5(1)	=	=	=	=	=	=
=	==	=	6(1)	=	=	=	=	=	=
-	-	-	7(1)	=	=	=	=	=	=
-	-	-	8(1)	=	=	=	=	=	=
-	-	-	9(1)	=	=	=	=	=	=
-	-	-	10(1)	=	=	=	=	=	=
-	-	-	11(1)	=	=	=	=	=	=
-	-	-	12(1)	=	=	=	=	=	=
=	==	=	13(1)	=	=	=	=	=	=
			14(1)						
114	11	90	Totals	3215	261	2200			
0.90			Hit Ratio				0.92		

One important figure in this display is the *Hit Ratio*. This number indicates what fraction of reads were satisfied by blocks found in memory. The hit ratio is calculated using the following formula:

$$\frac{(\text{cumulative logical reads} - \text{physical reads})}{\text{cumulative logical reads}}$$

In the example, this would be $(265 - 15) / 265$, or .94. The best possible value is 1, and if the value is less than .8 then additional performance tuning may be recommended.

The MONITOR ROLLBACK SEGMENT Display

MONITOR can be used to see the current status of rollback segments. A sample screen display is shown in Figure B-15.

FIGURE B-15
Sample MONITOR ROLLBACK
Display

RS	Rollback ID Segment	ORACLE Rollback Segment Monitor	Tue Nov 1 17:36:57
		Size Extents Active Xactions Write Rate Header	(bytes)
8	SYSTEM	175218 3 7 626 13 0	

When you have multiple rollback segments, the values in the Gets/sec column should be about equal for all rollback segments, to indicate that they are being accessed evenly.

If the value in the column Extents is high, then you may wish to create additional rollback segments, or increase the size of current segments.

A high value in the Waits/Sec column indicates contention for the headers of rollback segments. In this case, you should consider adding more rollback segments.

The MONITOR FILE I/O Display

The FILEIO display contains one row for every database file and summarizes the read and write activity on those files. Reads are performed by user processes, and all writes are performed by the background process DBWR. Figure B-16 shows the display.

If database files are added or dropped while this display is on the screen, you should redisplay the screen for an update.

FIGURE B-16
Sample MONITOR FILE I/O Display

ORACLE File I/O Monitor							Tue Nov 1 17:03:01
File Name	Request Rate Read/s	Request Rate Write/s	Batch Size blk/s/R	Response Time ms/Read	Response Time ms/Write	Total Blocks Read	Total Blocks Written
DISK\$DEV8:[U6KNLDUE.SHELLY]DATABASE1.ORA	11.40	14.40	1.00	1.00	0.00	8.00	18098 17874

If the value in column 2 or column 3, the *request rate of reads or writes per second*, is high for all files on a given drive, you should consider restructuring your database to allocate database files and tablespaces across additional disk drives.

The product of column 3 and column 5, *batch size of blocks written*, shows how much data was written to disk per second. If this value is high relative to the capacity of that disk, then you should consider changing the disk to one with a higher transfer rate.

To obtain the total of all physical reads or writes for a disk made by ORACLE, add the relevant columns for all files located on the same disk. Note that this display does not show I/O to other files that may be on the same disk.

You will only see values in column 6 and column 7 if you set the INIT.ORA parameter TIMED_STATISTICS to TRUE. The default value for this parameter is false, as there is some overhead associated with collecting the timing statistics.

ARCHIVE LOG

- Purpose** Start or stop automatic archiving of online redo log files, manually (explicitly) archive specified redo log files, or display the set of redo log files.
- Prerequisites** You must be connected to an open ORACLE database with DBA privilege and the database must be running in ARCHIVELOG mode.
- Syntax** ARCHIVE LOG [LIST] [STOP]
[START | n | NEXT | ALL] ['filespec']
- LIST** Use LIST to request a display that shows:
- range of redo log files to be archived
 - current log file's sequence number
 - current archive destination (specified by either the optional command text or by the INIT.ORA parameter ARCHIVE_LOG_DEST).

If you are using both ARCHIVELOG mode and automatic archiving, the display might appear like:

Database log mode	ARCHIVELOG
Automatic archival	ENABLED
Archive destination	DISK\$DEV9:[ORADEV] ARCH
Oldest online log sequence	30
Next log sequence to archive	33
Current log sequence number	33

Because the log sequence number of the current log and the next log to archive are the same, automatic archival has archived all logs up to the current one.

If you are using ARCHIVELOG but had disabled automatic archiving, the last three lines might look like:

Oldest online log sequence	30
Next log sequence to archive	30
Current log sequence number	33

If you are using NOARCHIVELOG mode, the "next log sequence to archive line" is suppressed.

The log sequence number increments every time LGWR begins to write to another redo log file; it does not indicate the number of logs being used. Every time an online redo log file is written anew, the contents are assigned a new log sequence number.

START	Enable automatic archiving. Starts the background process ARCH, which performs automatic archiving as required. If ARCH is started and <i>archive-destination</i> is supplied, then the new destination overrides the old.
	ARCH automatically starts on instance startup if the INIT.ORA parameter ARCHIVE_LOG_START is set to TRUE.
STOP	Disable automatic archival. If the instance is still in ARCHIVELOG mode then when all redo log files fill, database operation is suspended until the redo log file is archived (for example, until you enter the command ARCHIVE LOG START).
<i>n</i>	Any log sequence number which is still online. Causes archival of online redo log file log sequence number <i>n</i> . An error occurs if the log file cannot be found online or the sequence number is not valid. This option can be used to re-archive a log file.
NEXT	Manually archive the next online redo log file which has been filled but not yet archived.
ALL	Manually archive all filled, but not-yet-archived, online log files.
<i>filespec</i>	This filespec is used to specify the destination archive file in an operating system and installation-specific way. See the <i>Installation and User's Guide</i> for examples of specifying archive destinations. On many operating systems, multiple log files may be spooled to the same tape. Portions of the destination name may be used as a prefix for the archived files. If not specified in command line, the information is derived from the INIT.ORA parameter ARCHIVE_LOG_DEST. A destination specified in INIT.ORA may be temporarily overridden by specifying it with the ARCHIVE LOG command, but in order to make a destination the new default, a destination must be entered using the SQL*DBA command LOG ARCHIVE START <i>archive_dest</i> .
Notes	If an online redo log file fills and none are available for reuse, database operation is suspended. The condition can be resolved by archiving a log file.

Examples

```
ARCHIVE LOG START
ARCHIVE LOG STOP
ARCHIVE LOG 1001
```

CONNECT

Purpose	Connect to a database using the specified username.
Prerequisites	Only valid username/password combinations can successfully connect. The option CONNECT INTERNAL is a <i>system privileged</i> command.
Syntax	<code>CONNECT [{ username [/ password] } INTERNAL] [@instance-path]</code>
<i>username</i>	Any valid ORACLE username for the current database. May be a null string. If neither username nor INTERNAL are specified, SQL*DBA prompts for a username.
<i>password</i>	The password which corresponds to the specified username. May be a null string.
INTERNAL	Connect as username SYS via keyword INTERNAL. Should be used rarely and only by the DBA for certain maintenance purposes and requires operating system privileges. (See Chapter 2 for more information regarding connecting as INTERNAL.)
<i>instance-path</i>	A valid specification for an instance/database combination. If an instance is specified, it becomes the current instance for the duration of the connection. It does not become the default instance for subsequent connections.
Notes	If only the ORACLE username is specified, the password is requested using the prompt "Password:". The entered password is not echoed. CONNECT can be used without a DISCONNECT to connect to another username. The connect is always accompanied by the opening of a cursor.
Examples	<code>CONNECT</code> <ul style="list-style-type: none">• prompt appears for valid username and password• connects to database on the current default host

```
CONNECT scott/tiger
  • connects to database on current host as username SCOTT with
    password TIGER

CONNECT system
  • prompts for SYSTEM's password
  • connects to database on current host as DBA username
    SYSTEM

CONNECT INTERNAL
  • connects to database on current host as INTERNAL (does not
    request a password but requires the operating system privilege
    to connect as INTERNAL)

CONNECT scott/tiger@devvms
  • connects to database on node DEVVMS as username SCOTT
    with password TIGER
```

DISCONNECT

Purpose Disconnect from an ORACLE database.

Prerequisites You must be currently connected to a database.

Syntax DISCONNECT

Notes Upon disconnection, SQL*DBA reverts to the current default host.
Closes all open cursors and commits uncommitted transactions.

Examples DISCONNECT

EXIT

Purpose Exit SQL*DBA.

Prerequisites None.

Syntax EXIT

Notes Leaves SQL*DBA unconditionally, no matter what you are currently doing. Returns to the operating system prompt, if any. Commits uncommitted transactions.

Examples exit

HOST

Purpose Execute an operating system command without exiting SQL*DBA.

Prerequisites None.

Syntax `HOST [operating-system-command]`

Notes This command is particularly useful for listing current files of interest to the DBA, or for editing files to be run while in SQL*DBA. Your operating system may also support additional symbols or characters to invoke operating system commands.

If you omit the operating system command and enter simply HOST, you temporarily exit SQL*DBA on the assumption you want to run multiple operating system commands; to return to the current SQL*DBA session, you must use an operating system-specific entry. Refer to your *Installation and User's Guide* to find that entry.

Examples `HOST TYPE INIT.ORA`

MONITOR

Purpose	Display realtime information about locks, enqueues, processes and other database entities. (See the sample displays and descriptions of each display earlier in this appendix.)
Prerequisites	You must be connected to the database with a username that has SELECT access to the dynamic performance tables. SQL*DBA requires access to an appropriate CRT file to display the full screen information.
Syntax	<pre>MON[ITOR] [ROLLBACK FILEIO LATCH[ES]] [{ IO[S] LOCK[S] [ALL] PROCESS[ES] STATISTICS [CLASS] TABLE[S] USER[S] } [pid] [pid]]</pre>
	Descriptions and examples of the various displays can be found earlier in this appendix. If you enter MONITOR without any options, you will see a menu of these choices.
FILEIO	Displays a screen with one row for every database file, showing read and write activity on those files.
IO	Displays a histogram of the percent each ORACLE process contributes to the total logical and physical I/O during the specified period. This information indicates relative distributions, not precise counts of I/O operations.
LATCHES	Displays a screen summarizing all current latches.
LOCKS	Displays a screen indicating current ORACLE processes and the locks held or waited for.
PROCESSES	Displays one row of information for each process currently connected to the database.
ROLLBACK	Displays information regarding the size, status, and activity against all active rollback segments, with one row per rollback segment.
STATISTICS	Displays statistics regarding database use. The statistics are described earlier in this appendix.
TABLES	Displays names of tables referenced in currently parsed SQL statements, along with the table's address, the number of read and

update accesses, the process ID and the ORACLE username of the user accessing the table.

USERS Displays one screen of information for an individual active process. You can request information for a single process or a range of processes. Displays for the background processes vary slightly from those for user processes.

Notes Although for some options you may specify individual process numbers, by default the screens cycle through all processes. To explicitly request one screen summarizing all processes, use PID 0.

To exit from a display, use your system's interrupt mechanism. You will be returned to the SQLDBA prompt or to the menu.

Examples

```
MONITOR LATCH
MONITOR LOCK 10 20
MONITOR LOCK ALL
MONITOR PROCESS 11
```

RECOVER

Purpose Recover (roll forward) one or more tablespaces or the entire database.

Prerequisites You must be connected to ORACLE with DBA privilege. See Notes for additional prerequisites for different recovery situations.

Syntax RECOVER { DATABASE |
TABLESPACE *tablespace-name* [,tablespace-name] ... }

tablespace-name name of one or more tablespaces in the current database. You may recover up to 16 tablespaces in one statement.

Notes In order to perform media recovery on an entire database (all tablespaces), the database must be mounted exclusively and closed.

In order to perform media recovery on a tablespace, the database must be mounted and open, and the tablespace must be offline.

Before using the RESTORE command you must have restored good copies of the damaged database file(s) from a previous backup. Be sure you can access all archived and online redo log files dating back to when that backup was made.

When another log file is required during recovery, a prompt requests the name of a file which corresponds to the required log sequence number. You should enter the name of the file on disk to which you have restored the corresponding redo log, or you can enter a tape specification. Thus, you can restore directly from tape.

When recovery is done a completion status is returned.

To terminate media recovery, enter "CANCEL" as a file name.

Manual recovery to a previous point in time is possible and is documented in an online file named RECOVERY.DOC. This file also documents options for the RECOVER command that may be used in unusual circumstances.

Examples RECOVER DATABASE
RECOVER TABLESPACE TS_ONE, TS_TWO

REMARK

Purpose Enter a comment, typically in SQL files.

Prerequisites None.

Syntax REM [ARK]

Notes Primarily for batch use of SQL*DBA. The comment is ignored by SQL*DBA and by ORACLE.

REM must be the first non-blank character string in the line.

Examples: Following are examples of valid comments embedded in a SQL file:

```
REM      This command file is used to create a
REM      database. Edit it to fill in file names
REM      and sizes, and invoke it from SQLDBA.
REM
REM
CREATE DATABASE dbname ...
```

SET

Purpose Set or change characteristics of the current SQL*DBA session.

Prerequisites None.

Syntax

```
SET { INSTANCE [ instance-name ] |  
      ECHO [ ON | OFF ] |  
      TERMOUT [ ON | OFF ] |  
      TIMING [ ON | OFF ] |  
      STOPONERROR [ ON | OFF } |  
      NUMWIDTH [n] |  
      CHARWIDTH [n] |  
      LONGWIDTH [n] |  
      DATEWIDTH [n] |  
      MAXDATA [n] |  
      ARRAYSIZE [n] |  
      CYCLE seconds }
```

INSTANCE
instance-name Changes the current instance to the specified instance. Does not connect to a database. Invalid when already connected to a database. May be used to start, stop, or monitor a remote instance (with subsequent STARTUP, SHUTDOWN, or MONITOR commands).

Any commands preceding first use of SET INSTANCE will communicate to the default instance.

To reset the instance to the original, you can either enter SET INSTANCE with no *instance-name* or SET INSTANCE LOCAL. If you use SET INSTANCE DEFAULT, SQL*DBA searches for a remote instance named 'DEFAULT.'

ECHO	ON	Enables echoing of commands entered from command files.
	OFF	The default, disables echoing of commands.
TERMOUT	ON	The default, enables terminal output for SQL commands.
	OFF	Disables terminal output. Useful for preventing output to terminal when spooling output to files. Note that nothing will appear on the terminal until SET TERMOUT ON is used.
TIMING	ON	Displays parse, execute, and fetch times (CPU and elapsed) for each SQL statement executed.
	OFF	The default, does not display timing information.

STOPONERROR	ON	Indicates that if a command file is running and an error occurs, execution should terminate.
	OFF	Disables STOPONERROR.
NUMWIDTH [n]		Set the column display width for NUMBER data. If entered with no argument, returns the setting to the default. Default is 10.
CHARWIDTH [n]		Set the column display width for CHAR data. If entered with no argument, returns the setting to the default. Default is 80.
LONGWIDTH [n]		Set the column display width for LONG data. If entered with no argument, returns the setting to the default. Default is 80.
DATEWIDTH [n]		Set the column display width for DATE data. If entered with no argument, returns the setting to the default. Default is 9.
MAXDATA [n]		Set the maximum data size. Indicates the maximum data that can be received in a single fetch during a SELECT statement. The default is 4096 bytes (4K). The maximum (on VMS) is 64K.
ARRAYSIZE [n]		Sets the number of rows fetched at a time from the database. The default is 10 rows. The maximum (on VMS) is 100 rows.
CYCLE <i>seconds</i>		Set the cycle time in seconds for the MONITOR command. Minimum time is 1 second; maximum is 3600. Normally, a cycle time of at least 5 seconds is preferred.

Examples

```
SET INSTANCE DEVVMS
SET TIMING ON
SET CYCLE 10
SET LONGWIDTH 132
SET NUMWIDTH 20
SET CHARWIDTH 5
```

Both the following commands can be used to revert to the initial default host.

```
SET INSTANCE
SET INSTANCE LOCAL
```

SHOW

Purpose Show settings currently in effect.

Prerequisites None.

```
Syntax SHOW { ALL |  
          PARAMETERS [ string ] |  
          INSTANCE |  
          TIMING |  
          CYCLE |  
          CHARWIDTH |  
          LONGWIDTH  
          DATEWIDTH |  
          NUMWIDTH |  
          TERMOUT |  
          SGA |  
          SPOOL |  
          ECHO }
```

PARAMETERS Use this option to see the current values for one or many INIT.ORA parameters. You can use a string after the command to see a subset of parameters whose names include that string. For example, if you enter:

```
SHOW PARAMETERS COUNT
```

you would see:

NAME	TYPE	VALUE
CPU_COUNT	integer	0
DB_MULTIBLOCK_READ_COUNT	integer	8

Notes Most of these settings are set or changed via the SQL*DBA command SET. The exceptions are:

- SHOW SPOOL which is set via the command SPOOL
- SHOW SGA which returns the size of the SGA
- SHOW PARAMETERS which returns a list of all INIT.ORA parameters and their current values.

Until the command SET INSTANCE has been used, SHOW INSTANCE will return the value "local."

Examples SHOW TIMING

returns a display such as:

Parse	2.94 (Elapsed)	0.20 (CPU)
Execute/Fetch	13.17 (Elapsed)	2.78 (CPU)
Total	16.11	2.98

SHOW ALL

returns a display like:

Instance	local
Spool	OFF
Timing	OFF
Termout	ON
Echo	OFF
Stoponerror	OFF
cycle	5
Maxdata	4096
Arrayszie	5
Numwidth	10
Charwidth	80
Longwidth	80
Datewidth	9

SHOW SGA

returns a display like:

Total Shared Global Area	801280 bytes
	13896 bytes
Variable Size	369592 bytes
Database Buffers	409600 bytes
Redo Buffers	8192 bytes

SHUTDOWN

Purpose Shut down a currently running ORACLE instance, optionally closing and dismounting a database.

Prerequisites: *System privileged* command. You must not be currently connected to a database.

Syntax SHUTDOWN [NORMAL | IMMEDIATE | ABORT]

NORMAL The default, waits for currently connected users to disconnect from the database, prohibits further connects, and closes and dismounts the database. Finally, shuts down instance. Does not require instance recovery on next startup.

IMMEDIATE Does not wait for current calls to complete, prohibits further connects, and closes and dismounts the database. Finally, shuts down instance. Does not wait for connected users to disconnect. Does not require instance recovery on next startup.

ABORT Fastest possible shutdown. Does not wait for calls to complete or users to disconnect. Does not close or dismount database, but does shut down instance. Will require instance recovery on next startup. You must use this option if a background process terminates abnormally.

Notes SHUTDOWN with no arguments is equivalent to SHUTDOWN NORMAL.

Examples SQLDBA> SHUTDOWN

```
Database closed.  
Database dismounted.  
ORACLE instance shut down.
```

SPOOL

Purpose Enable or disable spooling of output to a specified file.

Prerequisites None.

Syntax SPOOL [*filename* | OFF]

filename Any valid name for a spool file. If not specified, the filetype or file extension is LOG. If a file by that name exists an error is generated.

OFF Close the previously opened spool file.

Notes The default filename is SQLDBA.LOG. To see whether you are currently spooling enter SHOW SPOOL.

Examples SPOOL DIMMICK

- creates a file named DIMMICK.LOG

STARTUP

Purpose	Start an ORACLE instance, with several options including mounting and opening a database.
Prerequisites	<i>System privileged command.</i> You must not be connected to a database.
Syntax	<code>STARTUP [option [option ...]]</code>
	where <i>option</i> is:
	<code>[DBA]</code> <code>[FORCE]</code> <code>[PFILE filespec]</code> <code>[EXCLUSIVE SHARED]</code> <code>[{ MOUNT OPEN } databasename]</code> <code>[NOMOUNT]</code>
DBA	Prohibits any ORACLE username without DBA privilege from connecting to the database.
FORCE	Shuts down the current ORACLE instance (if it is running) with SHUTDOWN mode ABORT before restarting it. If the current instance is running and FORCE is not specified, an error results. FORCE is useful while debugging and under abnormal circumstances -- should not normally be used.
PFILE <i>filespec</i>	Uses the specified parameter file while starting up. If no file is specified, the default is the INIT.ORA file found in its default location.
EXCLUSIVE	The default, signifies that the database can only be mounted and opened by the current instance (it cannot be opened simultaneously by multiple instances). Cannot be used with SHARED or NOMOUNT.
SHARED	Must be specified if the database is to be mounted by multiple instances concurrently. Cannot be used with EXCLUSIVE or NOMOUNT. Invalid if the INIT.ORA parameter SINGLE_PROCESS is set to TRUE.
MOUNT	Mounts a database but does not open it.
OPEN	Mounts and opens the specified database.

databasename Specifies the database name. If no database name is specified, then the database name is taken from the current INIT.ORA file parameter DB_NAME.

NOMOUNT Specifies that the database is not to be mounted upon instance startup. Cannot be used with EXCLUSIVE, MOUNT, or OPEN.

Notes No option may be used twice and some options are mutually exclusive. With no arguments, STARTUP will start the current instance (using the default database), mount, and open the database.

To start a remote instance, first use SET INSTANCE and then use the STARTUP command.

Examples

```
STARTUP  
STARTUP OPEN databasename EXCLUSIVE
```

- (commands are equivalent)
- starts an instance
- uses the standard INIT.ORA file
- mounts the default database in exclusive mode
- opens the database.

```
STARTUP SHARED  
STARTUP OPEN databasename SHARED
```

- (commands are equivalent)
- starts an instance
- uses the standard INIT.ORA file
- mounts the default database in shared mode
- opens the database.

```
STARTUP FORCE NOMOUNT DBA
```

- if the instance is currently running, uses SHUTDOWN ABORT to first shutdown the instance
- starts the instance
- no database is identified, mounted, or opened (the database name in INIT.ORA is ignored). EXCLUSIVE is assumed
- only DBAs are allowed to connect.

```
STARTUP PFILE=TESTPARM NOMOUNT
```

- starts an instance using parameters in file TESTPARM
- does not mount a database

- EXCLUSIVE and the database named in file TESTPARM are assumed, but not used.

```
STARTUP OPEN databasename PFILE MYINIT.ORA FORCE SHARED DBA
```

- if the instance is currently running, uses SHUTDOWN ABORT to first shutdown the instance
- starts the instance
- mounts in shared mode the database named in file MYINIT.ORA
- opens the database
- allows only DBAs to access the database.

```
STARTUP PFILE vendors
```

- Uses the vendors file instead of INIT.ORA.

```
STARTUP EXCLUSIVE NOMOUNT
```

- *invalid*, because EXCLUSIVE and NOMOUNT are mutually exclusive.

To startup an instance and mount but not open a database, you must use the following sequence of commands (the system's response is also shown):

```
SQLDBA> STARTUP NOMOUNT
ORACLE instance started.
```

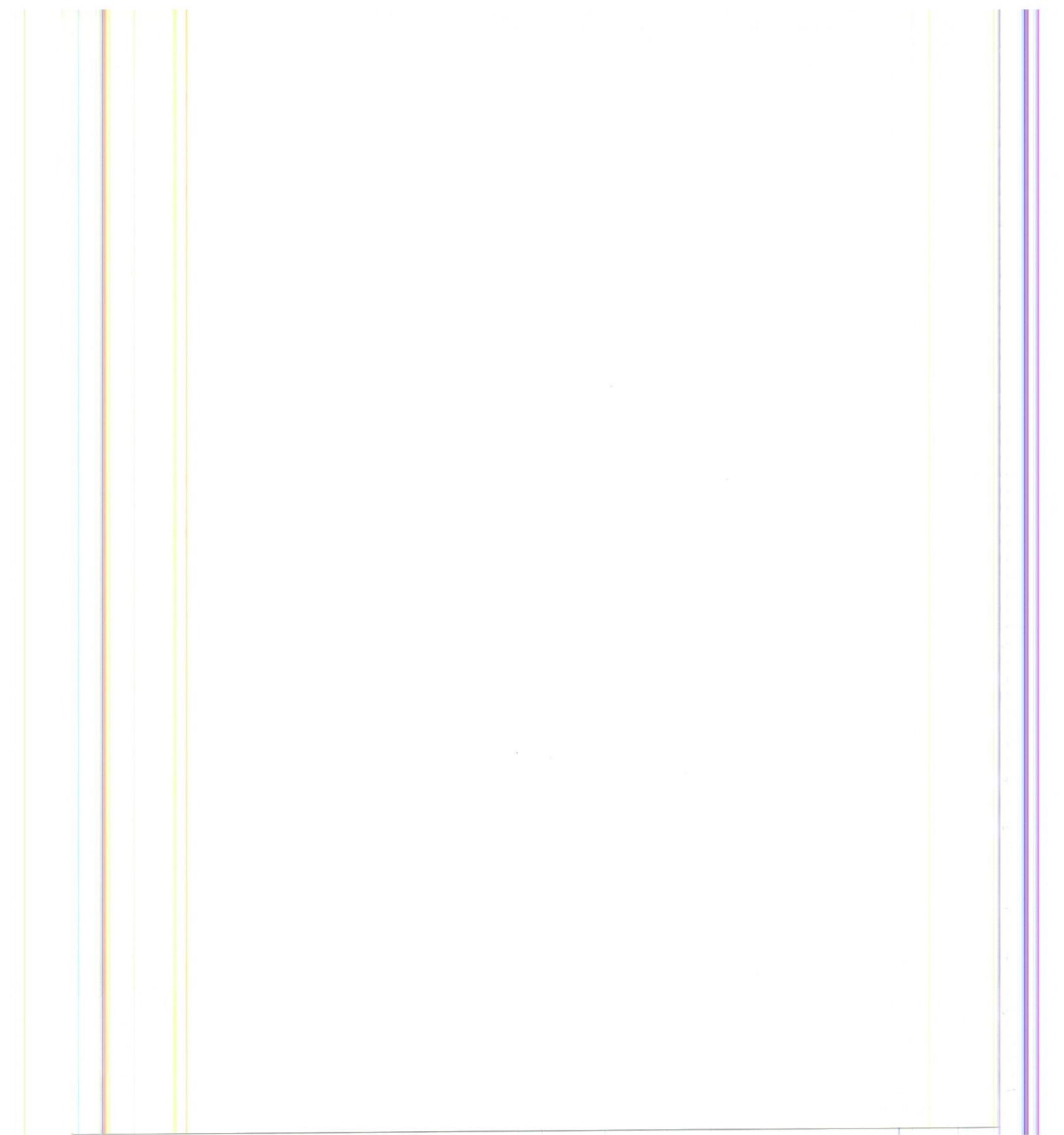
```
SQLDBA> CONNECT INTERNAL
Connected.
```

```
SQLDBA> ALTER DATABASE MOUNT;
Statement processed
```

At this point you could run a maintenance command, and then open the database, as shown in the following commands:

```
SQLDBA> ALTER DATABASE ARCHIVELOG;
Statement processed.
```

```
SQLDBA> ALTER DATABASE OPEN;
Statement processed.
```



C

SQL*DBA ERROR CODES AND MESSAGES

This appendix lists errors that may be returned when using SQL*DBA, with possible causes of the error and steps you may take to resolve the error.

The errors listed in this appendix are displayed with the prefix DBA, signifying SQL*DBA errors, or LCC, signifying errors in the command line. Errors for various Oracle products are documented in various places. The main source is the *ORACLE Error Codes and Messages* manual, but errors returned by particular products are documented in the manuals for those products.

DBA Errors

The following errors may be returned by SQL*DBA and appear with the prefix DBA.

300 internal error code, argument: [num]

Cause: You have encountered an internal error.

Action: Call Oracle Customer Support with the circumstances leading to the error and the complete set of error messages.

301 cannot SET INSTANCE while connected to a database

Cause: You used SET INSTANCE while you were currently connected to a database.

Action: DISCONNECT from the database before using SET INSTANCE if you wish to change the current instance.

302 not connected to a database

Cause: You must be connected to a database for the requested operation.

Action: CONNECT to the database using a valid username and password before retrying the operation.

303 cannot initialize the terminal

Cause: The .CRT file is invalid or could not be found. SQL*SDBA requires the .CRT file to display the MONITOR displays.

Action: Make sure SQL*DBA can access the desired .CRT file.

304 input file I/O error [num] - input aborted

Cause: A command file used as input to SQL*DBA is corrupt or invalid.

Action: Check the file before retrying the operation.

305 command size exceeds internal buffer size (num)

Cause: The size of the SQL statement exceeds SQL*DBA's buffer size.

Action: Shorten the SQL statement by removing extra blanks or by using intermediate statements or views, if necessary.

306 monitor cycle interval time out of range (1 - num)

Cause: You entered an invalid number for the cycle interval.

Action: Enter a number between 1 and 3600 for the cycle interval. The number indicates seconds.

307 cannot open spool file *text*

Cause: SQL*DBA tried to open a spool file after you entered SPOOL filename, but could not open the file. Possible causes are not enough disk space or inadequate privileges to create a file.

Action: Determine why SQL*DBA could not create a new file and retry.

308 no spool file opened

Cause: You entered SPOOL OFF, but you were not spooling currently and so there was no file to close.

Action: If you wish to capture session output, first use the SPOOL command to open a file, and then enter your commands before closing the file with SPOOL OFF.

309 cannot close spool file *text*

Cause: SPOOL OFF could not close the currently open spool file.

Action: Check for an operating system reason that the spool file could not be closed.

310 cannot open parameter file *text*

Cause: SQL*DBA cannot locate or open the file specified by the PFILE option, either because the file does not exist or SQL*DBA has insufficient privilege to open the file.

Action: Make sure that the file exists in a location expected by SQL*DBA and can be opened.

311 data size exceeds buffer size - data truncated

Cause: The results returned by a SQL query exceed the internal SQL*DBA buffer.

Action: Use the SET command to increase MAXDATA or decrease ARRAYSIZE.

312 invalid pid range *num num*

Cause: You entered an invalid range. Ranges must be specified using the lowest number first. If a valid range includes any currently active process identification numbers, the range will be accepted.

Action: Retry by entering a valid range, as in:

```
MONITOR PROCESSES 5 10..
```

313 no active processes to monitor

Cause: You specified process identification numbers, of which none correspond to currently active processes.

Action: Retry by entering process identification numbers that you know to be currently active.

315 cannot open command file *text*

Cause: SQL*DBA cannot locate the specified command file.

Action: Verify the file's name and SQL*DBA's access to it before retrying.

316 cannot startup while connected to a database

Cause: You attempted to startup an instance while you were connected to a database.

Action: If you wish to startup an instance, you must DISCONNECT from the database to which you are currently connected.

317 cannot shutdown while connected to a database

Cause: You attempted to shutdown an instance while you were connected to a database.

Action: If you wish to shutdown the instance, you must DISCONNECT from the database to which you are currently connected.

318 SQL*DBA command line error [*num*]

Cause: You made a syntax or typing error while entering a SQL*DBA command line.

Action: Check the syntax and try again.

320 terminal type name too long

Cause: You specified a terminal name that may or may not be valid, but is too long (exceeds 64 characters).

Action: Verify the name of the desired terminal and try again.

321 instance name too long

Cause: You specified an instance name that may or may not be valid, but is too long (exceeds 64 characters).

Action: Verify the name of the desired instance and try again.

322 total size of command line parameters exceeds buffer size

Cause: You entered too many command line arguments and the SQL*DBA buffer was exceeded.

Action: Reduce the number of command line arguments.

324 maximum number of tablespaces (*num*) exceeded - last *num* ignored

Cause: You specified too many tablespace names in the RECOVER TABLESPACE command. You may only specify up to 16 tablespace names.

Action: Reduce the number of tablespaces. If you want to recover more than 16 tablespaces, then use the RECOVER command multiple times.

325 pfile too large

Cause: The file you specified using PFILE is too large. (exceeds 8K).

Action: Reduce the size of the parameter file before specifying it again using PFILE.

328 Insufficient privilege for this display

Cause: You attempted to display a MONITOR display without sufficient privileges.

Action: Contact the DBA to obtain the required privileges.

329 Insufficient privilege for SHOW SGA

Cause: You attempted to use SHOW SGA without sufficient privileges.

Action: Contact the DBA to obtain the required privileges.

331 cannot allocate enough memory for SQL Buffer

Cause: There is not enough memory for the current SQL buffer.

Action: Use the SET command to reduce MAXDATA.

333 db file(s) added after MONITOR FILEIO invoked

Cause: Since invoking MONITOR FILEIO a database file has been added, invalidating the screen.

Action: Re-invoke MONITOR FILEIO.

334 db file(s) dropped after MONITOR FILEIO invoked

Cause: Since invoking MONITOR FILEIO a database file has been dropped, invalidating the screen.

Action: Re-invoke MONITOR FILEIO.

336 insufficient privilege for SHOW PARAMETERS

Cause: You attempted to use SHOW PARAMETERS without sufficient privileges.

Action: Contact the DBA to obtain the required privileges.

LCC Errors

The following errors may be returned when you enter a command or startup an instance. They are prefixed with LCC:

100 internal error, argument num

Cause: You have encountered an internal error.

Action: Call Oracle Customer Support with the circumstances leading to the error and the complete set of error messages.

111 value not in legal range *text*

Cause: The value of the parameter is outside the valid range.

Action: Check the valid range and retry using a new value.

112 illegal integer radix specification *text*

Cause:

Action: Valid characters are 'd', 'h', 'D' and 'H'.

113 integer conversion error or negative integer *text*

Cause: A non-integer or negative integer was assigned to an integer parameter.

Action: Retry using a valid integer value.

114 illegal boolean response *text*

Cause: You entered a value other than TRUE or FALSE.

Action: Enter a boolean value (either TRUE or FALSE).

122 unrecognized keyword *text*

Cause: You have probably used an invalid or misspelled keyword.

Action: Reenter the line.

201 could not open specified filename *text*

Cause: The specified file does not exist.

Action: Check the name of the file, or create a file by that name, before retrying.

203 missing keyword *text*

Cause: A keyword is expected but none was found.

Action: Add a parameter keyword followed by an equal sign and a parameter value.

204 left parentheses and no parameter specified *text*

Cause: A parameter list was started but no parameter was specified.

Action: Add an appropriate list of values and close the parameter list.

205 unbalanced parentheses *text*

Cause: An odd number of parentheses was found, indicating that one is missing or one is extra.

Action: Reenter the statement using the correct number of parentheses.

206 positional parameter entered after keyword *text*

Cause: An equal sign is missing.

Action: Reenter the parameter specification using an equal sign between the parameter name and the value you wish to specify.

207 nested parentheses encountered *text*

Cause: Only one set of parentheses is permitted.

Action: Remove the nested parentheses and retry.

208 unexpected keyword in value list *text*

Cause: A keyword was found instead of a value.

Action: Insert an appropriate value for the keyword.

209 missing value for keyword at end of string *text*

Cause: A keyword is specified with no value.

Action: Specify a valid value for the keyword.

210 illegal assignment operator *text*

Cause: You used a symbol other than an equal sign to assign a value to a parameter.

Action: Retry using an equal sign following the parameter name.

211 unexpected delimiter *text*

Cause: An invalid delimiter was found between values.

Action: A comma or a space is a valid delimiter.

212 runaway quoted string *text*

Cause: The parameter value was quoted on the left side but not on the right.

Action: Balance the quote marks.

215 parameter files nested too deep *text*

Cause: Too many parameter files have been nested. The maximum number of files that may be nested is 3.

Action: Reduce the number of nested parameter files.

217 failure while processing file parameter

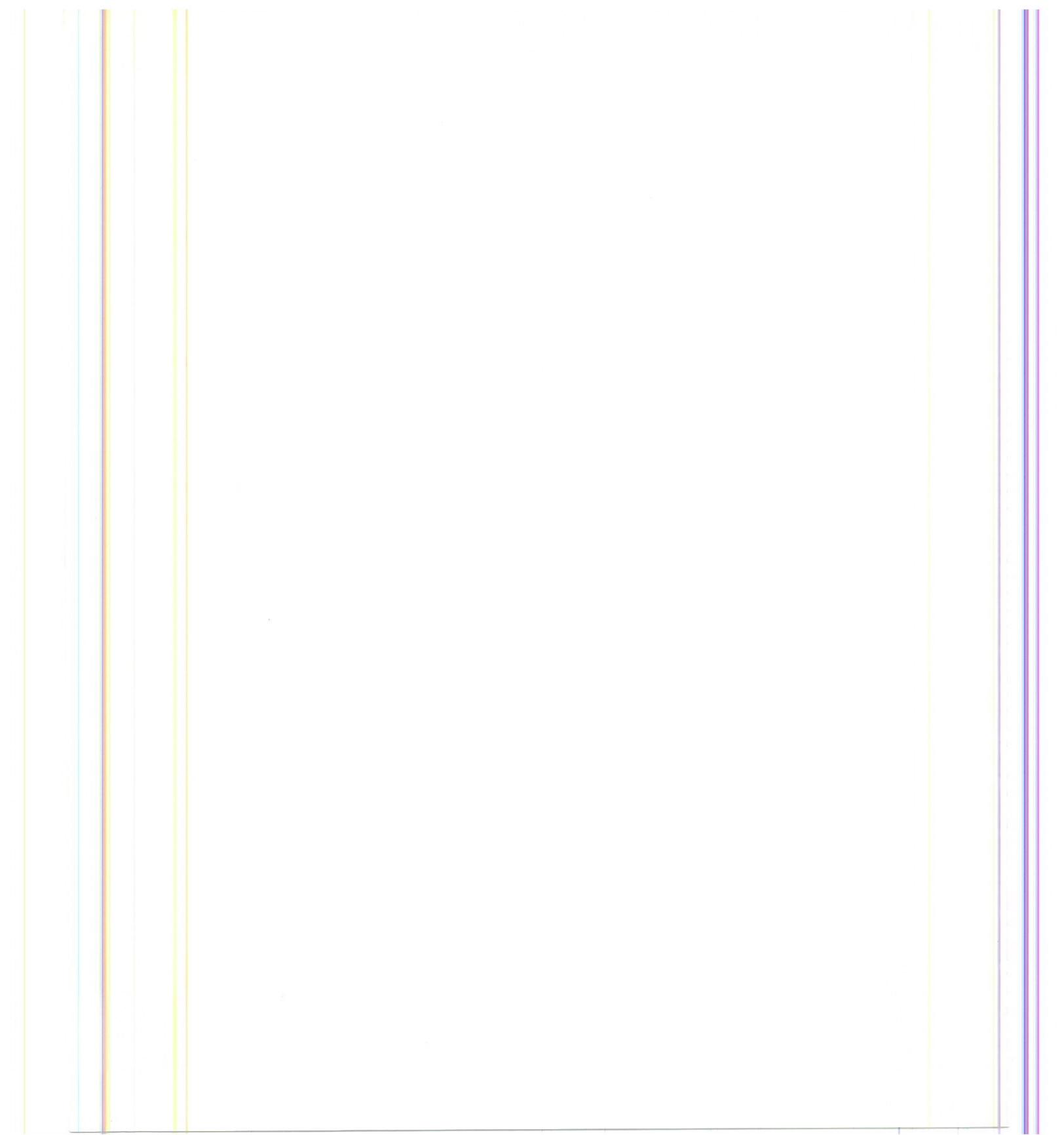
Cause: An error occurred while processing the specified file parameter.

Action: Look for other error messages with additional information.

218 error in file *text*

Cause: An error occurred in the particular file.

Action: Refer to the other messages for the specific error.



D

THE INIT.ORA PARAMETERS

This appendix contains detailed descriptions of the database startup parameters. The startup parameters are found in the INIT.ORA file. An INIT.ORA file must be accessible every time an instance is started.

Note: Though this guide refers to the parameter file as INIT.ORA, the actual name of the file may vary on the distribution media for your operating system. (For example, it may be in mixed or lower case, or it may have a logical name or a variation on the name INIT.ORA.) As DBA, you may also choose to use a file of another name (with the same contents and format). For simplicity, this manual consistently refers to the INIT.ORA file and the INIT.ORA parameters.

Note also that the expected location (where the RDBMS expects to find the file upon startup) is documented in the *Installation and User's Guide* for your operating system.

A general description and example of the INIT.ORA parameter file can be found in Chapter 14. A sample INIT.ORA is included on the ORACLE RDBMS distribution media for each operating system; this file is fine for initial use.

DBAs can use INIT.ORA parameters to:

- optimize performance, by adjusting memory structures (for example, increase the number of database buffers in memory)
- set some database-wide defaults (for example, how much space is initially allocated for a context area when it is created)
- set database limits (for example, setting a maximum number of database users)
- specify names of files.

Many parameters can be "fine-tuned" to improve database performance. Some parameters should never be altered or only be altered under supervision by Oracle Corporation personnel.

Specifying Parameters in the INIT.ORA File

The INIT.ORA file is a short text file that contains a list of parameters and a value for each parameter, as in:

```
SAVEPOINTS = 5  
PROCESSES = 100
```

The following rules govern the specification of parameters in the INIT.ORA file:

- All parameters are optional.
- Only parameters and comments should appear in the INIT.ORA file.
- If a parameter is not included in the INIT.ORA file, then the parameter's default value will be in effect.
- Parameters can be specified in any order.
- Case (upper or lower) is only significant (in filenames) if case is significant on the host operating system. Refer to your *Installation and User's Guide* if you require more information.
- To enter several parameters on one line, use spaces between parameter names, as in

```
PROCESSES = 100 SAVEPOINTS = 5 OPEN_CURSORS = 10
```

- For some parameters (such as INIT_SQL_FILES), multiple values can be entered by separating the values by commas and enclosing them in parentheses, as in:

```
INIT_SQL_FILES = (SQL.BSQ, CATALOG.SQL, ENROLL_USERS.SQL)
```

- A backslash (\) can be used to indicate continuation of the parameter specification.
- The keyword IFILE can be used to embed another file, which is expected to be in the same format as the INIT.ORA file. Up to three levels of nesting are allowed. The last value specified for a parameter overrides previous values.
- A pound sign (#) can be used to start a comment; the rest of the line will be ignored.

To change a parameter's value, edit the INIT.ORA file anytime an instance is shutdown. The next time the instance is started, the new parameter values in the updated INIT.ORA file are used.

Displaying Current Parameter Values

To see the current settings for INIT.ORA parameters, use the SQL*DBA command:

```
SQLDBA> SHOW PARAMETERS
```

This displays all parameters in alphabetical order, with their current value. If you enter a text string, as in

```
SQLDBA> SHOW PARAMETERS BLOCK
```

you will see a display for all parameters having BLOCK in their name. If you display all the parameters, you may wish to use the SPOOL command to write the output to a file.

You can also see the values by including the keyword LIST in the INIT.ORA file. Then when you start an instance using that parameter file, you see a display showing the current settings for the parameters.

Groups of Parameters

INIT.ORA parameters can be grouped in several different ways. For example, there are parameters that:

- set database-wide limits
- set user or process limits
- name files or directories required by a database system
- set limits on database resources
- affect performance (these are called *variable parameters*).

Perhaps the group of parameters of most interest to DBAs is the set of variable parameters, used primarily for improving database performance.

You may never need to worry about some parameters:

- Some parameters should never be altered except when you are instructed to do so by Oracle Corporation to resolve a problem.
- *Dependent parameters* should not be altered, as their values are automatically calculated by ORACLE.

Variable Parameters

Of the INIT.ORA parameters, variable parameters offer the most potential for improving system performance.

Some variable parameters set capacity limits. For example, if PROCESSES is 10, an 11th user process will not be able to connect to the database.

Some variable parameters affect performance but do not impose absolute limits; for example, a higher value for DB_BLOCK_BUFFERS usually improves performance, but a lower value does not prevent work, while it may slow it.

Increasing the values of variable parameters may improve your system's performance, but will also increase the size of the SGA. A larger SGA can improve database performance up to a point. In virtual memory operating systems, an SGA which is too large may degrade performance if it is swapped in and out of memory. Operating system parameters that control virtual memory working areas should be set with the size of the SGA in mind.

The most useful parameters in performance tuning are DB_BLOCK_BUFFERS and LOG_BUFFER. The dictionary cache parameters also have some effect on performance.

Dictionary Cache Parameters (with Prefix "DC")

Parameter names beginning with "DC" are *dictionary cache parameters*. An example is DC_TABLES. These parameters refer to caches in the SGA which temporarily hold entries from the base data dictionary tables. Entries are held in the caches for a very short time while the dictionary tables are used for modification or parsing. After that, the entries remain in the cache until the space is needed for newer entries.

These parameter values set the maximum number of entries that the resource may have in the SGA at any one time. Thus, these entries are memory resident. When this maximum is reached, the least recently used entry is replaced by the new entry (and the old entry is available from disk).

For example, if the DC_TABLES parameter is set to 10 in INIT.ORA, the SGA can store 10 table definitions. When the 11th table is accessed, one of the old table definitions is replaced by the definition for the 11th

table. If three users are using the table SCOTT.EMP, only one entry is required for the table.

These parameters have some effect on performance. Though it is not necessary to have enough entries for all simultaneous users, the cache sizes should be large enough to hold a working set of the most frequently accessed objects, so that the caches will not need to be constantly refreshed. An adequate cache size will improve performance of parsing SQL statements.

Global Cache Parameters (with Prefix "GC")

Parameters with the prefix "GC" are primarily for shared disk systems. An example is GC_SEGMENTS. The prefix "GC" stands for Global Cache. Their settings determine the size of the global collection of locks that protect the database buffers on all instances. The settings you choose will have an effect on the use of certain operating system resources. For additional information refer to the *Installation and User's Guide* for your operating system.

Operating System Dependent Parameters

For some parameters the valid ranges or values depend upon the host operating system; this is denoted in the default column as O/S dependent. For example, the parameter DB_BLOCK_BUFFERS indicates the *number* of data buffers in main memory; the *size* of those buffers, however, is system-dependent. Refer to the *Installation and User's Guide* for your particular operating system to see operating system specific information on the INIT.ORA parameters.

Dependent Parameters

Some parameters are noted as *dependent*. This means that their values are derived from values used for other parameters. Dependent parameters have no default except one derived from the default value of the other parameters. Normally you should not alter values for dependent parameters, but if you do, your value will override the calculated value.

When Parameters Are Set Incorrectly

Some parameters have a minimum setting below which an ORACLE instance will not start, whereas if the values are too low for other parameters, ORACLE may perform badly but will still run.

You may see error messages indicating that a parameter is too low or that you have reached the maximum for some resource.

Frequently you can wait a bit and retry the operation when the system is not as busy. If a message occurs repeatedly, you should shut the instance down, adjust the relevant INIT.ORA parameter, and restart the instance.

Individual Parameter Descriptions

Descriptions of the individual INIT.ORA parameters follow in alphabetical order. Parameter names are shown in this appendix and this manual in upper case for readability. See the *Installation and User's Guides* to see how your operating system expects to read the file and the parameters.

Most parameter values specified in INIT.ORA are global (on a database-wide basis), not per user, unless otherwise specified. When shown, VAX/VMS numbers are indicated for an example of the range of values.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
#			Used to signify a comment
\			Used to signify a continuation line will follow on the next line.
LIST			If used, current parameter values display upon system startup.
AUDIT_TRAIL -DOS False	false	yes	true/false
			Enables or disables the writing of rows to the audit trail. Auditing is disabled by default, if the value is FALSE, or if the parameter is not present. Auditing is enabled if parameter is set to TRUE. The SQL AUDIT statements can be used regardless of the setting. For more information on auditing refer to Chapters 17 and 18.
BACKGROUND_DUMP_DEST -DOS \	O/S dependent	yes	valid local pathname, directory, or disk
			The location (directory, disk, file) where debugging trace files for the background processes (LGWR, DBWR, DBRD, ARCH, SMON, and PMON) are written during the course of ORACLE operation. Only relevant for multi-user systems.
CALLS -DOS 12	51 calls	no	
			The number of recursive system calls. The distributed value is normally sufficient and should not be altered.
CLEANUP_ROLLBACK_ENTRIES -DOS 20	20 entries	yes	
			The number of rollback entries to process in a single cleanup pass during transaction cleanup. Normally will not need modification.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
CONTEXT_AREA <i>Dos 4096</i>	4096 bytes	yes	1024 - 131072
	The initial size in bytes of a new context area for which no size has been specified. The actual maximum on a given operating system may vary.		
CONTEXT_INCR <i>Dos 4096</i>	4096 bytes	yes	1024 - 32768
	The number of bytes by which a context area will grow each time it requires more space. The actual maximum on a given operating system may vary. The context area is described in Chapter 8.		
CONTROL_FILES	O/S dependent	yes	1 - 8
	One or more names of control files, separated by commas. Oracle Corporation recommends using multiple files on different devices. See Chapter 3 for a description of control files.		
CPU_COUNT <i>Dos 0</i>	0	yes	
	The number of CPUs for this instance. This number may be automatically updated on some operating systems. <u>Use 0</u> for a single processor system.		
DB_BLOCK_BUFFERS <i>Dos 500</i>	32 buffers	yes	4 - 65535
	The number of database blocks cached in memory in the SGA (one block equals one buffer). This parameter is the most significant determinant of the SGA size and database performance. The advantage of a higher value is that when a user needs a database block that block is more likely to be in memory, thus reducing I/O.		
	The size of each buffer is equal to the size of the parameter DB_BLOCK_SIZE.		
DB_BLOCK_CACHE_PROTECT <i>Dos False</i>	false	no	
	A debug mode in which the database blocks are protected. This parameter should not be altered except at the direction of Oracle personnel.		
DB_BLOCK_HASH_BUCKETS <i>Dos 125</i>	dependent	no	1 - 65521
	The size of the hash table used to find buffers in the buffer pool. Calculated from DB_BLOCK_BUFFERS; normally will not need modification. If set, the number should be prime and should not be 2.		

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
<code>DB_BLOCK_MAX_CLEAN_PCT</code>	33 %	yes	1% - 100 %
<p>Note: This parameter is obsolete in versions 6.0.27 and later.</p> <p>The percent of the cache that will be "clean" (free buffers) after each pass through the cache made by DBWR at the request of a user process (including buffers that were clean at the beginning of DBWR's pass).</p> <p>Higher values result in DBWR cleaning more buffers in less frequently occurring sessions (each time it runs, it cleans more blocks). This requires less overhead, but increases the risk of writing buffers that will be required again in the near future. You should increase the value when updates are affecting many different blocks. If value is too high, excessive I/O will occur. An increase in the value results in an decrease in the statistics <i>free buffer waits</i> and <i>DBWR free needed</i> (desirable).</p> <p>Lower values result in fewer buffers being cleaned at more frequent intervals, and a decreased possibility of writing newer buffers.</p> <p>It does not make sense to set <code>DB_BLOCK_MAX_CLEAN_PCT</code> to less than <code>DB_BLOCK_MAX_MOD_PCT</code>; in that case, the user process would spend time detecting that certain buffers are modified, but upon posting, DBWR would not be required to clean them.</p>			
<code>DB_BLOCK_MAX_MOD_PCT</code>	30 %	yes	1% - 100%
<p>Note: This parameter is obsolete in versions 6.0.27 and later.</p> <p>The percent of modified buffers in the cache needed to trigger writing by the DBWR background process in order to free buffers for use by user processes. If, while looking for a free buffer, a user process determines that this percent of buffers (or more) is modified, it notifies DBWR to begin writing buffers. This value should be lower than <code>DB_BLOCK_MAX_SCAN_PCT</code>.</p> <p>Higher values result in less frequent posting to DBWR, but searching of the LRU list is more CPU intensive and newer buffers are more likely to be reused prematurely. An increase in the value results in an increase in the statistics <i>free buffers inspected</i> and a decrease in <i>DBWR buffers scanned</i> (desirable).</p> <p>Lower values will post DBWR more often but reduce the search on the LRU list. An decrease in the value results in an decrease in the statistic <i>free buffers inspected</i> (desirable).</p>			

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
DB_BLOCK_MAX_SCAN_PCT	35 %	yes	1% - 100 %
Note: This parameter is obsolete in versions 6.0.27 and later.			
The maximum percent of the buffer cache that a user process will scan when looking for a free buffer. This value should normally be higher than DB_BLOCK_MAX_MOD_PCT.			
Increase this value when you can't write fast enough. If you increase it too high, the user processes will block DBWR.			
Decrease this value if the statistic <i>free buffers inspected</i> is high. If the value is too low, user processes will wait needlessly.			
DB_BLOCK_MULTIPLE_HASHCHAIN_LATCHES			
Dos : True	true	no	false/true
If set to TRUE, one latch is used per hash chain; if set to FALSE, one latch is used for all changes.			
DB_BLOCK_SIZE	O/S dependent	yes	512 - 8096 O/S dependent
Dos : 1024			
The size in bytes of ORACLE database blocks. Typical values are 2048 and 4096. The value for DB_BLOCK_SIZE in effect at CREATE DATABASE time determines the size of the blocks; at all other times the value must be set to the original value. See Chapter 3 for a discussion of choosing the block size and your <i>Installation and User's Guide</i> for the default value for your operating system.			
DB_BLOCK_TIMEOUT_WRITE_PCT	20 %	yes	0% - 100%
Note: This parameter is obsolete in versions 6.0.27 and later.			
Whenever DBWR is notified due to a timeout, it will clean approximately this percent of the cache (regardless of whether DB_BLOCK_MAX_CLN_PCT is already satisfied).			
Higher values cause the buffers to be written more quickly when the system is idle and may speed access to blocks when the usage picks up, but increase the chance that blocks will be written more than necessary.			
Lower values cause the cache to be written more slowly and are desirable when system resource use should be minimized (emphasizing throughput over response time). Lower values are also preferable when more transactions are queries than updates.			
Zero is an acceptable value and suppresses timeouts.			

	Default value	OK to change?	Range of values
DB_BLOCK_WRITE_BATCH <i>Ans 8</i>	8 blocks	yes	1 - 128 blocks
			Used for multi-block I/O, this is the number of blocks DBWR gives at one time to the operating system for writing.
			Higher values increase the ability of the operating system to write blocks on different disks in parallel, to take advantage of disk elevator algorithms, and to write adjacent blocks in a single I/O. However, all blocks in a batch are pinned until they are all written, so a higher value increases the chances a user process will have to wait to modify a block which is in a batch being written. You can increase this parameter until you notice the statistics <i>write complete waits</i> and <i>write wait time</i> increasing.
			If the operating system only supports one write to a device at a time, then the value should be between 2 and 8, where the only purpose of batching is to save some latching, unlatching, and other overhead.
DB_FILE_MULTIBLOCK_READ_COUNT	O/S dependent	yes	O/S dependent
			Used for multi-block I/O, this is the maximum number of blocks read when doing I/O during a sequential scan. The default is a function of DB_BLOCK_BUFFERS and PROCESSES. Values in the range of 4 to 16 or even 32 are reasonable. The actual maximums vary by operating system; VMS limits it to 65 Kbytes, or 31 2Kbyte blocks per read.
DB_FILE_SIMULTANEOUS_WRITES <i>4</i>	4	yes	24
			The number of simultaneous writes ("batches") for each database file when written by DBWR.
			If the operating system only supports one write per device, and cannot combine writes to adjacent blocks, then the value should be 1. If the system has a good elevator algorithm and/or channel programs, then increase the value a bit. Values over 5 are not usually useful.
			Though the value has no maximum, since DBWR does writes in groups of DB_BLOCK_WRITE_BATCH blocks, it is not useful to use a value larger than DB_BLOCK_WRITE_BATCH.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
DB_FILES <i>Dos 32</i>	32 files	yes	1 - MAXDATAFILES
	The maximum number of database files that can be opened at runtime for this database. Reduce only if you need SGA space and have fewer database files. You can increase the value by shutting and restarting the instance. Higher values slightly increase the size of the control file.		
	DB_FILES is similar to the MAXDATAFILES argument for the CREATE DATABASE statement used to set the absolute number of files at database creation.		
DB_HANDLES <i>Dos 8</i>	dependent	no	
	The maximum number of buffers simultaneously accessed by all user processes. Calculated as 4 times the number of PROCESSES.		
DB_HANDLES_CACHED <i>Dos 2</i>	2	no	
	The number of buffer handles per user process.		
DB_NAME <i>oracle</i>	null	yes	
	A database identifier of eight characters or less. Although its use is optional, it is recommended that it be set before invoking CREATE DATABASE, and then referenced in that statement. If specified, it must correspond to the name specified in the CREATE DATABASE statement. If not specified, then a database name must appear on either the STARTUP or ALTER DATABASE MOUNT command line.		
DC_COLUMN_GRANTS <i>Dos 50</i>	50	yes	1 - unlimited
	The number of entries in the column grant cache. Allow one for each column in use having grants, not one for each grant.		
DC_COLUMNS <i>Dos 500</i>	300	yes	150 - unlimited
	The number of entries in the column descriptions cache. The value should equal at least the highest number of columns in any single table. For best performance, value should be close to the number of columns referenced by all concurrent users at any one time.		
DC_CONSTRAINT_DEFS <i>Dos 200</i>	200	yes	unlimited
	The number of entries in the constraint definition cache.		

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
DC_CONSTRAINTS <i>Dos 180</i>	150	yes	unlimited
			The number of entries in the constraint cache.
DC_FILES <i>Dos 25</i>	25	yes	1 - unlimited
			The number of entries in the file descriptions cache. Allow one for each file associated with tablespaces in use.
DC_FREE_EXTENTS <i>Dos 50</i>	50	yes	5 - unlimited
			The number of entries in the free extent descriptions cache (which caches definitions of individual extents of free space, no matter what size).
DC_INDEXES <i>Dos 50</i>	50	yes	20 - unlimited
			The number of entries in the index descriptions cache.
DC_OBJECT_IDS <i>Dos 50</i>	50	yes	1 - unlimited
			The number of entries in the object identifier cache.
DC_OBJECTS <i>Dos 100</i>	100	yes	50 - unlimited
			The number of entries in the object descriptions cache (which caches names of database objects such as tables, clusters, indexes, sequences, synonyms, and views). The value for this parameter affects performance at parse time. For best performance, the value should be as large as the number of objects in use by all users at any one time.
DC_ROLLBACK_SEGMENTS <i>Dos 25</i>	25	yes	2 - unlimited
			The number of entries in the rollback segment description cache.
DC_SEGMENTS <i>Dos 50</i>	50	yes	50 - unlimited
			The number of entries in the segment description cache. Allow one for each table, cluster, index, and rollback segment in use at any one time.
DC_SEQUENCE_GRANTS <i>Dos 20</i>	20	yes	2 - unlimited
			The number of grants on sequences that can be cached.
DC_SEQUENCES <i>Dos 20</i>	20	yes	2 - unlimited
			The number of entries in the sequence description cache.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
DC_SYNONYMS <i>DOS 50</i>	50	yes	2 - unlimited
			The number of entries in the synonym description cache.
DC_TABLE_GRANTS <i>DOS 50</i>	50	yes	2 - unlimited
			The number of entries in the table grant cache. One entry is needed for each table with grants, not for each grant.
DC_TABLES <i>DOS 100</i>	100	yes	30 - unlimited
			The number of entries in the table/cluster/view descriptions cache.
DC_TABLESPACE_QUOTAS <i>DOS 25</i>	25	yes	1 - unlimited
			Number of entries in the tablespace quota cache, which caches individual user's quotas for each tablespace. If 10 users were simultaneously creating tables in 2 tablespaces, 20 entries would be required.
DC_TABLESPACES <i>DOS 25</i>	25	yes	2 - unlimited
			The number of entries in the tablespace description cache.
DC_USED_EXTENTS <i>DOS 50</i>	50	yes	50 - unlimited
			The number of entries in the used extents description cache.
DC_USERNAMES <i>DOS 50</i>	50	yes	1 - unlimited
			The number of entries in the username cache.
DC_USERS <i>DOS 50</i>	50	yes	1 - unlimited
			The number of entries in the user description cache.
DDL_LOCKS <i>DOS = 250 5x 50</i>	5 * SESSIONS	yes	20 - unlimited
			The maximum number of parse locks held simultaneously. There must be one for each table referenced in all currently open cursors. Value is global. For example, if 3 users are modifying data in one table, then 3 entries are required. If 3 users were modifying data in 2 tables, then 6 entries are required. For a description of DDL locks, see Chapter 12. The default value assumes that on average, open cursors reference 5 tables per session.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
DML_LOCKS <i>Dos 80 4x20</i>	4 * TRANSACTIONS	yes	20 - unlimited, 0
	The maximum number of DML locks — one for each table modified in a transaction. Value should equal the grand total of locks on tables referenced by all users. For example, if 3 users are modifying data in one table, then 3 entries would be required. If 3 users were modifying data in 2 tables, then 6 entries would be required. For a description of DML locks, see Chapter 12. The default value assumes an average of 4 tables referenced per transaction.		
	If set to zero (0) enqueues are disabled and performance is increased. However, you cannot use DROP TABLE or CREATE INDEX.		
ENQUEUE_HASH <i>Dos 20</i>	dependent	no	
	The length of the enqueue hash table. The value is derived from ENQUEUE_RESOURCES and should not be altered.		
ENQUEUE_LOCKS <i>Dos 30</i>	dependent	no	
	The number of individual locks owned by a process, other than DDL and DML. The value is derived from ENQUEUE_RESOURCES and should not need to be altered.		
ENQUEUE_RESOURCES <i>Dos 85</i>	dependent	yes	10 - 65,535
	The number of resources that can be locked by the lock manager. Allow one per resource, not per lock. This number should equal DML_LOCKS plus DDL_LOCKS plus overhead of about 20. The value is derived from PROCESSES and should be adequate. If many tables are used, it may be increased.		
EVENT <i>Dos ✓</i>	null	no	
	Control events while in debug mode. This parameter should not be altered except at the direction of Oracle personnel.		
FIXED_DATE <i>Dos.</i>	null	yes	
	Allows you to set a constant for SYSDATE in the usual form for ORACLE dates. Useful primarily for testing.		
FREE_LIST_INST <i>Dos Non</i>	1	yes	1 - # of instances
	Number of free list entries for instances, which is hashed over instances. The value should be equal to the number of instances (1 for single-instance) and no performance gain is realized by exceeding that value. The product of this parameter and FREE_LIST_PROC should be equal to or less than 12.		

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
FREE_LIST_PROC	4	yes	1 to 12
	<i>Dos Non</i>		Number of free list entries for processes. If any tables are expected to have very high INSERT activity, then this number should be increased. The product of this parameter and FREE_LIST_INST should be equal to or less than 12.
			Note: The following seven parameters apply only for shared disk systems (multiple instances sharing one database). See Chapter 21 for a description of shared disk systems.
GC_DB_LOCKS	dependent	yes	1 - unlimited
	<i>Nm</i>		Relevant only for shared disk systems, the maximum number of modified blocks that may be held in caches of all instances combined. The default is reasonable for systems of 1 or 2 instances; the value should be reset for systems with more instances. A reasonable value is: (sum of DB_BLOCK_BUFFERS for all instances) / 2
GC_ROLLBACK_LOCKS	20	yes	
	<i>Non</i>		Relevant only for shared disk systems, the number of simultaneous modified rollback segment blocks in all instances combined. The default is adequate for 1 or 2 instances but should be increased to 10 per instance for more instances.
GC_ROLLBACK_SEGMENTS	20	yes	
	<i>Dos 20</i>		Relevant only for shared disk systems, should be set to the total number of rollback segments system-wide.
GC_SAVE_ROLLBACK_LOCKS	20	yes	
	<i>Dos 20</i>		Relevant only for shared disk systems, the number of simultaneous modified deferred rollback segment blocks in all instances combined. The default is adequate for 1 or 2 instances but should be increased to 10 per instance for more instances.
GC_SEGMENTS	10	yes	
	<i>Dos 10</i>		Relevant only for shared disk systems, the total number of simultaneous segments system-wide which may have had space management activities performed. The default is adequate for 1 or 2 instances but should be increased to 10 per instance for more instances.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
GC_SORT_LOCKS	dependent	no	Relevant only for shared disk systems, the total number of sort locks.
GC_TABLESPACES <i>DOS 5</i>	5	yes	Relevant only for shared disk systems, the total number of tablespaces that can be brought from offline to online (or online to offline) concurrently.
IFILE <i>DOS</i>	null	yes	Used to embed another file within the current INIT.ORA file. Should be followed by the file's name, as in IFILE=TESTPARM.ORA. Up to three levels of nesting are allowed.
INIT_SQL_FILES <i>%ORACLE_HOME%</i> <i>\% — DBS\%</i>	O/S dependent	yes	Specifies the name of one or more files containing SQL statements to be executed when the database is created. The list should be enclosed in parentheses and the filenames separated by commas, as in: INIT_SQL_FILES = (SQL.BSQ, CATALOG.ORA, SITEDE.D.SQL)
			This parameter should always specify SQL.BSQ first. Then other site-specific files can be specified if desired. The default is operating system dependent and should not be altered, except to add additional files.
INSTANCES <i>Non</i>	16	yes	1 - 255
			Relevant only for shared disk systems, the current maximum number of instances which can simultaneously open the database. This parameter is different from the MAXINSTANCES option to CREATE DATABASE; MAXINSTANCES determines the absolute maximum number of instances. INSTANCES can temporarily reduce the limit set by MAXINSTANCES but it cannot increase it. The value can be changed between startups, as long as it is less than MAXINSTANCES.
LANGUAGE	'AMERICAN_AMERICA.US7ASCII'	yes	A character string that defines the national language operation of the database. This parameter actually defines three arguments in the format language_territory.char-set. The <i>language</i> argument specifies the language to be used for ORACLE messages, day and month names used by TO_CHAR and TO_DATE,

and the default date format. Examples of supported languages are American, French, German, Spanish.

The *territory* argument specifies the name of the territory whose day and week numbering convention is to be used. The value determines whether the week numbers calculated with the TO_CHAR function will use the ISO standard and whether "day 1" is Sunday or Monday. Supported territories include America, France, Germany, and Spain.

The *char_set* argument specifies the default character set used for ORACLE messages and processing (in particular, by the functions UPPER, LOWER, INITCAP and CONVERT), and the default collating sequence used by ORDER BY. Supported character sets (which may vary by operating system) are shown in the following table.

<i>Character Set Name</i>	<i>Description</i>
US7ASCII	U.S. 7-bit ASCII
WE8HP	West European (Multinational) 8-bit for HP
WE8DEC	West European 8-bit for DEC
WE8IBMPC	West European 8-bit for IBM PC/compatibles
F7DEC	French 7-bit for DEC
D7DEC	German 7-bit for DEC

See Appendix F for more information on the use and effects of this parameter and for more details on National Language Support.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
LOG_ALLOCATION	200 blocks <i>Dos 1000</i>	yes	Meaningful primarily for shared disk systems, indicates the number of redo log file blocks (independent of block size) allocated to an instance each time it requires more space in a current online log file. Chapter 21 describes using redo log files in shared disk systems and setting optimal allocation sizes.
			For a non-shared disk system, because every allocation goes to the same instance, it is most efficient to set the allocation size equal to (or greater than) the size of the redo log file.
			The number of such allocations made for an individual instance is reflected in the statistic <i>chunk allocations</i> .
LOG_ARCHIVE_DEST	O/S dependent <i>Dos oui</i>	yes	Applicable only if using the redo log in ARCHIVELOG mode and archiving is automatic (if LOG_ARCHIVE_START is TRUE). Use a text string to indicate the location and root of the disk file or tape device when archiving log files. Chapter 15 describes using automatic archiving and specifying a destination.
LOG_ARCHIVE_START	false <i>Dos oui</i>	yes	false/true
			Applicable only if using redo log in ARCHIVELOG mode, indicates whether archiving should be automatic or manual when the instance is started. TRUE indicates that archiving will be automatic. FALSE indicates that the DBA will archive filled redo log files. Chapter 15 describes both automatic and manual archiving.
			If you are creating the database in ARCHIVELOG mode, then you should set this parameter to TRUE (normally databases are created in NOARCHIVELOG mode and then altered to ARCHIVELOG mode after creation).
LOG_BUFFER	O/S dependent <i>40%</i>	yes	O/S dependent - unlimited
			The number of bytes allocated to the redo log buffers in the SGA. In general, larger values reduce redo log file I/O, particularly if transactions are long or numerous. In a busy system, the value 65536 or higher would not be unreasonable.
			The default is set to 4 times the maximum database block size for the host operating system (on VAX/VMS the default is 8192).

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
LOG_BUFFERS_DEBUG	False	no	true/false
	Ordinarily this should be set to false. Upon the request of Oracle Support personnel it may be set to true; in this case buffers are filled with known data to help locate problems with LGWR.		
LOG_CHECKPOINT_INTERVAL <i>O/S dependent</i>		yes	20 - no maximum
<i>Pos 104</i>	The number of newly filled redo log file blocks (units are operating system blocks, not ORACLE blocks) needed to trigger a checkpoint. Regardless of this value, a checkpoint will always occur when switching from one online log file to another. The size may exceed the actual size of any redo log file in which case checkpoints occur only when switching logs.		
	The number of checkpoints that have occurred for a given instance is shown in the statistic <i>dbwr checkpoints</i> .		
LOG_DEBUG_MULTI_INSTANCE	false	no	true/false
<i>No 7</i>	Ordinarily this should be set to false. Upon the request of Oracle Support personnel it may be set to true; in this case the log code will run in multi-instance mode even if the rest of the system does not.		
LOG_ENTRY_PREBUILD_THRESHOLD	0 bytes	yes	0 - unlimited
<i>O</i>	For multi-processor systems, it is sometimes beneficial to increase this parameter, which is the maximum number of bytes of redo to gather together before the copy to the log buffer. Single processor systems should keep the value at 0.		
	For systems experiencing latch contention, and that have fast processors with good memory-to-memory copy algorithms, an increase in this value will pre-build log entries, resulting in less time that the copy latch is held. Systems experiencing memory contention should not increase this parameter.		

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
LOG_FILES	16 255	yes	2 - 255
			The maximum number of redo log files which can be opened at runtime for this database. Reduce only if you need SGA space and have fewer log files. You can increase the value by shutting and restarting the instance. Higher values slightly increase the size of the control file.
			Different from the MAXLOGFILES argument for the CREATE DATABASE statement which sets the absolute number of log files at database creation.
LOG_IO_SIZE	0	O/S dependent (usually 0)	no
			The maximum number of blocks to write at one time to the online redo log file. The value is calculated based on the value for LOG_BUFFER and the block size of the log and is recalculated for every log file.
			You should leave this value set to 0, which indicates that the value will be automatically determined for each log file.
LOG_SIMULTANEOUS_COPIES	1 or n 1	yes	1 - n
			The maximum number of redo buffer copy latches able to write log buffers at any given time (where <i>n</i> equals the number of processors). If not specified, the number will default to the number of CPUs. For single-processor systems, set to 0. If this parameter is set to 0, then the parameter LOG_SMALL_ENTRY_MAX_SIZE is ignored.
LOG_SMALL_ENTRY_MAX_SIZE	800	O/S dependent	yes
			The size in bytes of the largest copy to the log buffers that may occur under the redo allocation latch, without obtaining the redo buffer copy latch. (On VMS the default is approximately 800 bytes.)
			If the value for LOG_SIMULTANEOUS_COPIES is 0, then this parameter is ignored (all writes are "small" and are made without the copy latch).
MAX_DUMP_FILE_SIZE	500 blocks 500	yes	
			Maximum size in operating system blocks of any trace files written. Set this limit if you are concerned that dump files may take up too much space.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
MESSAGES <i>20</i>	dependent	no	
	The maximum number of interprocess message buffers allocated for communication between user processes and background processes. The value is calculated based on PROCESSES and should not normally be altered.		
NLS_SORT <i>oui</i>	false	yes	false/true
	If the value is TRUE, sorting (as in the collating sequence for 'ORDER BY' queries) is based on the character set indicated by the value for parameter LANGUAGE.		
	If the value is FALSE, then sorting is based on the numeric value of characters (a binary sort that requires more system overhead).		
OPEN_CURSORS <i>100</i>	50	yes	5-255
	The maximum number of open cursors per user process, resulting in a greater or smaller address/memory space used by each process. The allocation per cursor is not large, only requiring space for the cursor descriptors.		
OPEN_LINKS <i>4</i>	4	yes	0 - 255
	The maximum number of concurrent open connections to remote databases per user process. Value should equal or exceed the number of databases referred to in any single SQL statement that references multiple databases, so that all the databases can be open in order to execute the statement. Value should be increased if many different databases are accessed over time. Thus, if queries alternately access databases A, B, and C and OPEN_LINKS is set to 2, time would be spent waiting while one connection was broken and another made.		
	This parameter refers only to connections used for distributed queries. Direct connections to a remote database specified as an application connects are not counted.		
	If set to 0, then no distributed queries are allowed.		
PRE_PAGE_SGA	false	yes	false/true
	If set to TRUE, then all pages of the SGA are paged into each user's working set. This parameter is VMS-specific and is primarily useful during benchmark testing.		

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
PROCESSES	25	yes	5 to O/S dependent
	25	For multi-process operation, the maximum number of operating system user processes that can simultaneously connect to ORACLE. Should include up to 5 for the background processes plus 1 for logon, so a value of 20 would permit 14 or 15 concurrent users.	
ROLLBACK_SEGMENTS	null	yes	
	oui 100	One or more rollback segments to be specifically allocated to this instance. An instance will always acquire all the segments indicated by this parameter, even if the number of segments exceeds the minimum number of segments calculated to be required by the instance (using TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT). The segments can be created with or without the keyword PUBLIC.	
	Never specify the SYSTEM rollback segment as a value for this parameter.		
ROW_CACHE_BUFFER_SIZE	200	no	
	200	The size in bytes of the row cache circular buffer.	
ROW_CACHE_ENQUEUES	100	yes	
	500	The number of "concurrent row cache managed enqueues". In CREATE TABLE, each NOT NULL constraint entry requires 1.	
ROW_CACHE_INSTANCE_LOCKS	100	no	
	100	The number of row cache instance locks.	
ROW_CACHE_MULTI_INSTANCE	true	no	false/true
	Num	Only applicable for shared disk systems. If value is TRUE then the row cache will support multiple instances.	
ROW_CACHE_PCT_FREE	10	no	
	10	Percent of free row cache parent objects.	

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
ROW_LOCKING	see description	yes	ALWAYS/INTENT
	INTENT		For ORACLE with the transaction processing option, the default is ALWAYS, which means that only row locks are acquired when a table is updated.
			For ORACLE V6 without the transaction processing option, the default and only valid value is INTENT, which means that only row locks are used on a SELECT FOR UPDATE but at update time table locks are acquired.
			The setting should be the same for all instances sharing a database.
SAVEPOINTS	5	yes	0-255
	5		The maximum number of simultaneous savepoints active per user process. See Chapter 11 for a description of savepoints.
SCN_INCREMENT	2048	no	
	2048		The global <i>system commit number</i> is incremented by this value when the current range is exhausted. This value should not normally be altered.
SCN_THRESHOLD	512	no	
	512		The range for the global system commit number is considered exhausted when this many are left. This value should not normally be altered.
SEQUENCE_CACHE_ENTRIES	10	yes	10- 32,000
	10		The number of sequence numbers that can be cached in the SGA for immediate access. This cache is managed on a least recently used basis, so if a request is made for a sequence that is not in the cache, and there are no free entries, the oldest one will be deleted and replaced with the newly requested one. Highest concurrency is achieved when this value is set to the highest possible number of sequences that will be used on an instance at one time.
			Each entry requires approximately 90 bytes in the SGA; on a shared disk system each entry requires approximately 110 bytes.
			Sequences that are created with the NOCACHE option do not reside in this cache, as they have to be written through to the dictionary on every use.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
SEQUENCE_CACHE_HASH_BUCKETS	7	yes	1-32,000
			The number of buckets (at about 8 bytes per bucket) is used to speed lookup for newly requested sequences in the cache; the cache is arranged as a hash table, and when a process makes its first request for a sequence, it looks for it in this table. It is not meaningful to have this larger than SEQUENCE_CACHE_ENTRIES. This value should be prime, and the system will automatically use the smallest prime greater than or equal to the value specified.
SERIALIZABLE	false	yes	false/true
	False		If TRUE, then queries acquire read locks, thus preventing any update of objects read until the transaction containing the query is committed. Provides degree three consistency at a considerable cost in concurrency.
SESSIONS	(1.1 * PROCESSES)	yes	
	7		The total number of user and system sessions. The default number allows for recursive sessions.
SINGLE_PROCESS	false	yes	false/true
	False		Determines whether a database is brought up in single user (same as single-process) or multi-user (multi-process) mode. TRUE indicates single user and FALSE indicates multi-user. For a description of single-process and multi-process ORACLE systems, refer to Chapter 9. Shared disk systems must set this value to FALSE.
SORT_AREA_SIZE	O/S dependent	yes	
	9216		The size in bytes of real memory which can reasonably be expected to be available for sorting. For example, on a VAX with 8 megabytes of real memory, of which you think 1/8 might be available to sorting processes, and you expect to have about 4 sorts occurring at once, you might set this parameter to $8192/8/4 = 256K$. As a rule, a larger size will only improve the efficiency of large sorts.
			The default is usually fine for most database operations. Only if very large indexes are being created might you want to adjust this parameter. For example, if one process is doing all database access, as in a full database import, then an increased value for this parameter may speed the import, particularly the CREATE INDEX statements.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
SORT_READ_FAC 5	O/S dependent	no	The multi-block read factor for sorting.
SORT_SPACEMAP_SIZE 286	O/S dependent	yes	The size in bytes of the sort spacemap in context area. Only if you have very large indexes should you adjust this parameter. A sort automatically increases its spacemap if necessary, but it won't necessarily do so when it will make best use of disk storage. The sort will make optimal use of disk storage if SORT_SPCMAP_SIZE is set to: $[(\text{total-sort-bytes}) / (\text{sort-area-size})] + 64$ where <i>total-sort-bytes</i> is: $(\text{number-of-records}) * [\sum-\text{of-average-column-sizes} + (2 * \text{number of col})]$ where columns include the SELECT list for the ORDER BY, the SELECT list for the GROUP BY, and the key list for CREATE INDEX. Also include 10 bytes for ROWID for CREATE INDEX and GROUP BY or ORDER BY columns not mentioned in the SELECT list for these cases.
TIMED_STATISTICS True	false	yes	false/true By default (when set to FALSE) the SQL*DBA statistics related to time (from the buffer manager) always are zero and the RDBMS can avoid the overhead of requesting the time from the operating system. To turn on statistics, set the value to TRUE. Should normally be set to FALSE.
TRANSACTIONS 20	1.1 * PROCESSES	yes	The maximum number of concurrent transactions. Increases size of SGA and the number of rollback segments allocated. The default is slightly higher than the number of processes to allow for recursive transactions. See Chapter 15 for a discussion of transactions and rollback segments.
TRANSACTIONS_PER_ROLLBACK_SEGMENT 30	20	yes	The number of concurrent transactions allowed per rollback segment. Thus, the number of rollback segments acquired at startup would be TRANSACTIONS divided by the value for this parameter. For example, if TRANSACTIONS were 101 and this parameter were 10, then the number of rollback segments acquired would be the rounded result of 101/10, or 11.

	<i>Default value</i>	<i>OK to change?</i>	<i>Range of values</i>
USER_DUMP_DEST <i>(u)</i>	O/S dependent	yes	The location (directory, file, disk) where debugging trace files from a user process are written.
USE_ROW_ENQUEUES <i>R/Se</i>	true	yes	true/false
	By default, in-memory enqueues are obtained for row locks so they are granted in the right order. This parameter controls access to rows by requiring all updaters to first get row enqueues on the rows to be modified. This prevents extra work and retries in the case of rows with high contention, and reduces I/O on multi-instance systems. If there is no row contention and you wish to achieve a small performance gain, you can set this parameter to FALSE, eliminating enqueues for these situations. You might do this when using enqueues would use too much SGA memory or require too much overhead. However, if there is row contention and the value is FALSE, I/O characteristics may suffer. The setting of this parameter does not affect the holding of the locks or the integrity of data and order of changes; it affects only the order in which locks on rows are obtained by different processes. This parameter must have the same value on all instances sharing a database.		
USER_SESSIONS <i>S</i>	1	no	1
	The maximum number of sessions per user process. Currently limited to 1 session per user process.		

E

DATA DICTIONARY TABLES

This appendix contains descriptions of the data dictionary tables and views. To see the current data dictionary on your system, query the view DDICTIONARY.

This appendix contains the following information:

- names and descriptions of the data dictionary views
- names and description of their columns
- names of the dynamic performance tables and columns.

The Data Dictionary Views

The following is an alphabetical reference of the data dictionary views accessible to users and DBAs. This information is also available directly in the data dictionary COMMENTS columns.

ACCESSIBLE_COLUMNS	Columns of all tables, views and clusters
DATA_DEFAULT	Default value for the column
DEFAULT_LENGTH	Length of default value for the column
COLUMN_ID	Sequence number of the column as created
NULLABLE	Does column allow NULL values?
DATA_SCALE	Digits to right of decimal point in a number
DATA_PRECISION	Length: digits (NUMBER) or characters (CHAR,RAW)
DATA_LENGTH	Length of the column in bytes
DATA_TYPE	Datatype of the column
COLUMN_NAME	Column name
TABLE_NAME	Table, view or cluster name
OWNER	Owner of the table, view or cluster
ACCESSIBLE_TABLES	Tables and views accessible to the user
TABLE_TYPE	Type of the object
TABLE_NAME	Name of the object
OWNER	Owner of the object
ALL_CATALOG	All tables, views, synonyms, sequences accessible to the user
TABLE_TYPE	Type of the object
TABLE_NAME	Name of the object
OWNER	Owner of the object
ALL_COL_COMMENTS	Comments on columns of accessible tables and views
COMMENTS	Comment on the column
COLUMN_NAME	Name of the column
TABLE_NAME	Name of the object
OWNER	Owner of the object
ALL_COL_GRANTS_MADE	Grants on columns for which the user is owner or grantor
CREATED	Timestamp for the grant
REFERENCES_PRIV	Permission to make REFERENCES to the column?

	UPDATE_PRIV	Permission to UPDATE the column?
	GRANTOR	Name of the user who performed the grant
	COLUMN_NAME	Name of the column
	TABLE_NAME	Name of the object
	OWNER	Username of the owner of the object
	GRANTEE	Name of the user to whom access was granted
ALL_COL_GRANTS_REC'D	Grants on columns for which the user or PUBLIC is the grantee	
	CREATED	Timestamp for the grant
	REFERENCES_PRIV	Permission to make REFERENCES to the column?
	UPDATE_PRIV	Permission to UPDATE the column?
	GRANTOR	Name of the user who performed the grant
	COLUMN_NAME	Name of the column
	TABLE_NAME	Name of the object
	OWNER	Username of the owner of the object
	GRANTEE	Name of the user to whom access was granted
ALL_DB_LINKS	Database links accessible to the user	
	CREATED	Creation time of the database link
	HOST	SQL*Net string for connect
	USERNAME	Name of user to log on as
	DB_LINK	Name of the database link
	OWNER	
ALL_DEF_AUDIT_OPTS	Auditing options for newly created objects	
	UPD	Auditing UPDATE WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	SEL	Auditing SELECT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	REN	Auditing RENAME WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	LOC	Auditing LOCK WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	INS	Auditing INSERT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	IND	Auditing INDEX WHENEVER SUCCESSFUL / UNSUCCESSFUL.

	GRA	Auditing GRANT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	DEL	Auditing DELETE WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	COM	Auditing COMMENT WHENEVER SUCCESSFUL / UNSUCCESSFUL
	AUD	Auditing AUDIT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	ALT	Auditing ALTER WHENEVER SUCCESSFUL / UNSUCCESSFUL.
ALL_INDEXES	Descriptions of indexes on tables accessible to the user	
	PCT_INCREASE	Percentage increase in extent size
	MAX_EXTENTS	Maximum number of extents allowed in the segment
	MIN_EXTENTS	Minimum number of extents allowed in the segment
	NEXT_EXTENT	Size of secondary extents
	INITIAL_EXTENT	Size of the initial extent
	MAX_TRANS	Maximum number of transactions
	INI_TRANS	Initial number of transactions
	TABLESPACE_NAME	Name of the tablespace containing the index
	UNIQUENESS	Uniqueness status of the index: "UNIQUE" or "NONUNIQUE"
	TABLE_TYPE	Type of the indexed object
	TABLE_NAME	Name of the indexed object
	TABLE_OWNER	Owner of the indexed object
	INDEX_NAME	Name of the index
	OWNER	Username of the owner of the index
ALL_IND_COLUMNS	Columns comprising indexes on accessible tables	
	COLUMN_LENGTH	Indexed length of the column
	COLUMN_POSITION	Position of column within index
	COLUMN_NAME	Column name
	TABLE_NAME	Table or cluster name
	TABLE_OWNER	Table or cluster owner
	INDEX_NAME	Index name
	INDEX_OWNER	Index owner

ALL_OBJECTS	Objects accessible to the user
MODIFIED	Timestamp for the last DDL change to the object
CREATED	Timestamp for the creation of the object
OBJECT_TYPE	Type of the object
OBJECT_ID	Object number of the object
OBJECT_NAME	Name of the object
OWNER	Username of the owner of the object
ALLSEQUENCES	Description of sequences accessible to the user
LAST_NUMBER	Last sequence number written to disk
CACHE_SIZE	Number of sequence numbers to cache
ORDER_FLAG	Are sequence numbers generated in order?
CYCLE_FLAG	Does sequence wrap around on reaching limit?
INCREMENT_BY	Value by which sequence is incremented
MAX_VALUE	Maximum value of the sequence
MIN_VALUE	Minimum value of the sequence
SEQUENCE_NAME	SEQUENCE name
SEQUENCE_OWNER	Name of the owner of the sequence
ALL_SYNONYMS	All synonyms accessible to the user
DB_LINK	Name of the database link referenced in a remote synonym
TABLE_NAME	Name of the object referenced by the synonym
TABLE_OWNER	Owner of the object referenced by the synonym
SYNONYM_NAME	Name of the synonym
OWNER	Owner of the synonym
ALL_TABLES	Description of tables accessible to the user
BACKED_UP	Has table been backed up since last modification?
PCT_INCREASE	Percentage increase in extent size
MAX_EXTENTS	Maximum number of extents allowed in the segment
MIN_EXTENTS	Minimum number of extents allowed in the segment
NEXT_EXTENT	Size of secondary extents in bytes
INITIAL_EXTENT	Size of the initial extent in bytes

	MAX_TRANS	Maximum number of transactions
	INI_TRANS	Initial number of transactions
	PCT_USED	Minimum percentage of used space in a block
	PCT_FREE	Minimum percentage of free space in a block
	CLUSTER_NAME	Name of the cluster, if any, to which the table belongs
	TABLESPACE_NAME	Name of the tablespace containing the table
	TABLE_NAME	Name of the table
	OWNER	Owner of the table
ALL_TAB_COMMENTS	Comments on tables and views accessible to the user	
	COMMENTS	Comment on the object
	TABLE_TYPE	Type of the object
	TABLE_NAME	Name of the object
	OWNER	Owner of the object
ALL_TAB_GRANTS_MADE	User's grants and grants on user's objects	
	CREATED	Timestamp for the grant
	INDEX_PRIV	Permission to CREATE/DROP INDEX on the object?
	ALTER_PRIV	Permission to ALTER the object?
	REFERENCES_PRIV	Permission to make REFERENCES to the object?
	UPDATE_PRIV	Permission to UPDATE the object?
	DELETE_PRIV	Permission to DELETE from the object?
	INSERT_PRIV	Permission to INSERT into the object?
	SELECT_PRIV	Permission to SELECT from the object?
	GRANTOR	Name of the user who performed the grant
	TABLE_NAME	Name of the object
	OWNER	Owner of the object
	GRANTEE	Name of the user to whom access was granted
ALL_TAB_GRANTS_REC'D	Grants on objects for which the user or PUBLIC is the grantee	
	CREATED	Timestamp for the grant
	INDEX_PRIV	Permission to create/drop an INDEX on the object?
	ALTER_PRIV	Permission to ALTER the object?

	REFERENCES_PRIV	Permission to make REFERENCES to the object?
	UPDATE_PRIV	Permission to UPDATE the object?
	DELETE_PRIV	Permission to DELETE from the object?
	INSERT_PRIV	Permission to INSERT into the object?
	SELECT_PRIV	Permission to SELECT from the object?
	GRANTOR	Name of the user who performed the grant
	TABLE_NAME	Name of the object
	OWNER	Owner of the object
	GRANTEE	Name of the user to whom access was granted
ALL_USERS	Information about all users of the database	
	CREATED	User creation date
	USER_ID	ID number of the user
	USERNAME	Name of the user
ALL_VIEWS	Text of views accessible to the user	
	VIEW_NAME	Name of the view
	OWNER	Owner of the view
AUDIT_ACTIONS	Description table for audit trail action type codes	
	NAME	Name of the type of audit trail action
	ACTION	Numeric audit trail action type code
COLUMN_PRIVILEGES	Grants on columns for which the user is the grantor, grantee, or owner, or PUBLIC	
	CREATED	Timestamp for the grant
	REFERENCES_PRIV	Permission to make REFERENCES to the column?
	UPDATE_PRIV	Permission to UPDATE the column?
	GRANTOR	Name of the user who performed the grant
	COLUMN_NAME	Name of the column
	TABLE_NAME	Name of the object
	OWNER	Username of the owner of the object
	GRANTEE	Name of the user to whom access was granted

CONSTRAINT_COLUMNS	Information about accessible columns in constraint definitions	
POSITION		Ordinal position of column in definition
COLUMN_NAME		Name associated with column specified in the constraint definition
CONSTRAINT_NAME		Name associated with the constraint definition
OWNER		Owner of the constraint definition
CONSTRAINT_DEFS	Constraint definitions on accessible tables	
R_CONSTRAINT_NAME		Name of unique constraint definition for referenced table
R_OWNER		Owner of table used in referential constraint
SEARCH_CONDITION		Text of search condition for table check
TABLE_NAME		Name associated with table with constraint definition
CONSTRAINT_TYPE		
CONSTRAINT_NAME		Name associated with constraint definition
OWNER		Owner of the table
DBA_AUDIT_DBA		
RETURNCODE		Oracle error code generated by the action. Zero if the action succeeded.
STATEMENTID		Numeric ID for each statement run (a statement may cause many actions).
ENTRYID		Numeric ID for each audit trail entry in the session.
SESSIONID		Numeric ID for each Oracle session.
GRANTEE		The name of the grantee specified in a GRANT/REVOKE statement.
RESOURCE_PRIV		Y or - for RESOURCE privilege did or did not appear in GRANT/REVOKE statement.
DBA_PRIV		Y or - for DBA privilege did or did not appear in GRANT/REVOKE statement.
CONNECT_PRIV		Y or - for CONNECT privilege did or did not appear in GRANT/REVOKE statement.
ACTION_NAME		Name of the action type corresponding to the numeric code in ACTION.
ACTION		Numeric action type code. The corresponding name of the action type (CREATE TAB)
OBJ_NAME		Name of the object affected by the action.
OWNER		Creator of object affected by the action.

	TIMESTAMP	Timestamp for the creation of the audit trail entry. (Timestamp for the user's l
	TERMINAL	Identifier for the user's terminal.
	USERHOST	Numeric instance ID for the Oracle instance from which the user is accessing the
	USERNAME	Name (not ID number) of the user whose actions were audited.
DBA_AUDIT_EXISTS		
	RETURNCODE	Oracle error code generated by the action. Zero if the action succeeded.
	STATEMENTID	Numeric ID for each statement run (a statement may cause many actions).
	ENTRYID	Numeric ID for each audit trail entry in the session.
	SESSIONID	Numeric ID for each Oracle session.
	GRANTEE	The name of the grantee specified in a GRANT/REVOKE statement.
	PRIVILEGE	Privileges granted/revoked by a GRANT/REVOKE statement.
	NEW_NAME	The new name of an object renamed by a RENAME statement.
	ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION.
	OBJ_NAME	Name of the object affected by the action.
	OWNER	Intended creator of the non-existent object.
	TIMESTAMP	Timestamp for the creation of the audit trail entry.
	TERMINAL	Identifier for the user's terminal.
	USERHOST	Numeric instance ID for the Oracle instance from which the user is accessing the
	USERNAME	Name (not ID number) of the user whose actions were audited.
DBA_CATALOG		
	TABLE_TYPE	Type of the object
	TABLE_NAME	Name of the object
	OWNER	Owner of the object

DBA_CLUSTERS	PCT_INCREASE MAX_EXTENTS MIN_EXTENTS NEXT_EXTENT INITIAL_EXTENT MAX_TRANS INI_TRANS KEY_SIZE PCT_USED PCT_FREE TABLESPACE_NAME CLUSTER_NAME OWNER	Percentage increase in extent size Maximum number of extents allowed in the segment Minimum number of extents allowed in the segment Size of secondary extents in bytes Size of the initial extent in bytes Maximum number of transactions Initial number of transactions Estimated size of cluster key plus associated rows Minimum percentage of used space in a block Minimum percentage of free space in a block Name of the tablespace containing the cluster Name of the cluster Owner of the cluster
DBA_CLU_COLUMNS	TAB_COLUMN_NAME TABLE_NAME CLU_COLUMN_NAME CLUSTER_NAME OWNER	Key column in the table Clustered table name Key column in the cluster Cluster name Owner of the cluster
DBA_COL_COMMENTS	COMMENTS COLUMN_NAME TABLE_NAME OWNER	Comment on the object Name of the column Name of the object Name of the owner of the object
DBA_COL_GRANTS	CREATED REFERENCES_PRIV UPDATE_PRIV GRANTOR COLUMN_NAME	Timestamp for the grant Permission to make REFERENCES to the column? Permission to UPDATE the column? Name of the user who performed the grant Name of the column

	TABLE_NAME	Name of the object
	OWNER	Username of the owner of the object
	GRANTEE	Name of the user to whom access was granted
DBA_CROSS_REFS		
	REF_DB_LINK	Database link of the referenced object
	REF_TABLE_NAME	Name of the referenced object
	REF_OWNER	Owner of the referenced object
	TABLE_TYPE	Type of the referencing object
	TABLE_NAME	Name of the referencing object
	OWNER	Owner of the referencing object
DBA_DATA_FILES		
	STATUS	File status: "INVALID" or "AVAILABLE"
	BLOCKS	Size of the file in ORACLE blocks
	BYTES	Size of the file in bytes
	TABLESPACE_NAME	Name of the tablespace to which the file belongs
	FILE_ID	ID of the database file
	FILE_NAME	Name of the database file
DBA_DB_LINKS		
	CREATED	Creation time of the database link
	HOST	SQL*Net string for connect
	PASSWORD	Password for logon
	USERNAME	Name of user to log on as
	DB_LINK	Name of the database link
	OWNER	
DBA_EXP_FILES		
	TIMESTAMP	Timestamp of the last export
	USER_NAME	Name of user who executed export
	FILE_NAME	Name of the export file
	EXP_VERSION	Version number of the last export
DBA_EXP_VERSION		
	EXP_VERSION	Version number of the last export session

DBA_EXTENTS	BLOCKS BYTES BLOCK_ID FILE_ID EXTENT_ID TABLESPACE_NAME	Size of the extent in ORACLE blocks Size of the extent in bytes Starting block number of the extent Name of the file containing the extent Extent number in the segment Name of the tablespace containing the extent
	SEGMENT_TYPE SEGMENT_NAME	Type of the segment Name of the segment associated with the extent
	OWNER	Owner of the segment associated with the extent
DBA_FREE_SPACE	BLOCKS BYTES BLOCK_ID FILE_ID TABLESPACE_NAME	Size of the extent in ORACLE blocks Size of the extent in bytes Starting block number of the extent ID number of the file containing the extent Name of the tablespace containing the extent
DBA_INDEXES	PCT_INCREASE MAX_EXTENTS MIN_EXTENTS NEXT_EXTENT INITIAL_EXTENT MAX_TRANS INI_TRANS TABLESPACE_NAME UNIQUENESS TABLE_TYPE TABLE_NAME TABLE_OWNER INDEX_NAME OWNER	Percentage increase in extent size Maximum number of extents allowed in the segment Minimum number of extents allowed in the segment Size of secondary extents Size of the initial extent Maximum number of transactions Initial number of transactions Name of the tablespace containing the index Uniqueness status of the index: "UNIQUE" or "NONUNIQUE" Type of the indexed object Name of the indexed object Owner of the indexed object Name of the index Username of the owner of the index

DBA_IND_COLUMNS	COLUMN_LENGTH COLUMN_POSITION COLUMN_NAME TABLE_NAME TABLE_OWNER INDEX_NAME INDEX_OWNER DBA_OBJECTSMODIFIED	Indexed length of the column Position of column within index Column name Table or cluster name Table or cluster owner Index name Index owner Timestamp for the last DDL change to the object
	CREATED OBJECT_TYPE OBJECT_ID OBJECT_NAME OWNER	Timestamp for the creation of the object Type of the object Object number of the object Name of the object Username of the owner of the object
DBA_ROLLBACK_SEGS	STATUS PCT_INCREASE MAX_EXTENTS MIN_EXTENTS NEXT_EXTENT INITIAL_EXTENT BLOCK_ID	Rollback segment status Percent increase for extent size Maximum number of extents Minimum number of extents Secondary extent size in bytes Initial extent size in bytes
	FILE_ID SEGMENT_ID TABLESPACE_NAME	ID number of the block containing the segment header ID number of the rollback segment Name of the tablespace containing the rollback segment
	OWNER SEGMENT_NAME	Owner of the rollback segment Name of the rollback segment
DBA_SEGMENTS	MAX_EXTENTS EXTENTS BLOCKS BYTES	Maximum number of extents allowed in the segment Number of extents allocated to the segment Size, in Oracle blocks, of the segment Size, in bytes, of the segment

	HEADER_BLOCK	ID of the block containing the segment header
	HEADER_FILE	ID of the file containing the segment header
	TABLESPACE_NAME	Name of the tablespace containing the segment
	SEGMENT_TYPE	Type of segment: "TABLE", "CLUSTER", "INDEX", "ROLLBACK", "DEFERRED ROLLBACK",
	SEGMENT_NAME	Name, if any, of the segment
	DBA_SEGMENTS	OWNER
		Username of the segment owner
DBA_SEQUENCES		
	LAST_NUMBER	Last sequence number written to disk
	CACHE_SIZE	Number of sequence numbers to cache
	ORDER_FLAG	Are sequence numbers generated in order?
	CYCLE_FLAG	Does sequence wraparound on reaching limit?
	INCREMENT_BY	Value by which sequence is incremented
	MAX_VALUE	Maximum value of the sequence
	MIN_VALUE	Minimum value of the sequence
	SEQUENCE_NAME	SEQUENCE name
	SEQUENCE_OWNER	Name of the owner of the sequence
DBA_SYNONYMS		
	DB_LINK	Name of the database link referenced in a remote synonym
	TABLE_NAME	Name of the object referenced by the synonym
	TABLE_OWNER	Owner of the object referenced by the synonym
	SYNONYM_NAME	Name of the synonym
	OWNER	Username of the owner of the synonym
DBA_SYS_AUDIT_OPTS		
	Describes current system auditing options	
	RESOURCE_ACTION	Auditing RESOURCE actions.
	NOT_EXISTS	Auditing actions on objects that do NOT EXIST.
	DBA_ACTION	Auditing actions requiring DBA privilege.
	CONNECT_ACTION	Auditing CONNECT/DISCONNECT actions.

DBA_TABLES

BACKED_UP	Has table been backed up since last modification?
PCT_INCREASE	Percentage increase in extent size
MAX_EXTENTS	Maximum number of extents allowed in the segment
MIN_EXTENTS	Minimum number of extents allowed in the segment
NEXT_EXTENT	Size of secondary extents in bytes
INITIAL_EXTENT	Size of the initial extent in bytes
MAX_TRANS	Maximum number of transactions
INI_TRANS	Initial umber of transactions
PCT_USED	Minimum percentage of used space in a block
PCT_FREE	Minimum percentage of free space in a block
CLUSTER_NAME	Name of the cluster, if any, to which the table belongs
TABLESPACE_NAME	Name of the tablespace containing the table
TABLE_NAME	Name of the table
OWNER	Owner of the table

DBA_TABLESPACES

STATUS	Tablespace status: "ONLINE" or "OFFLINE"
PCT_INCREASE	Default percent increase for extent size
MAX_EXTENTS	Default maximum number of extents
MIN_EXTENTS	Default minimum number of extents
NEXT_EXTENT	Default incremental extent size
INITIAL_EXTENT	Default initial extent size
TABLESPACE_NAME	Tablespace name

DBA_TAB_AUDIT_OPTS

UPD	Auditing UPDATE WHENEVER SUCCESSFUL / UNSUCCESSFUL.
SEL	Auditing SELECT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
REN	Auditing RENAME WHENEVER SUCCESSFUL / UNSUCCESSFUL.
LOC	Auditing LOCK WHENEVER SUCCESSFUL / UNSUCCESSFUL.

	INS	Auditing INSERT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	IND	Auditing INDEX WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	GRA	Auditing GRANT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	DEL	Auditing DELETE WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	COM	Auditing COMMENT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	AUD	Auditing AUDIT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	ALT	Auditing ALTER WHENEVER SUCCESSFUL / UNSUCCESSFUL.
DBA_TAB_COLUMNS	TABLE_TYPE	Type of the object.
DBA_TAB_COLUMNS	TABLE_NAME	Name of the object.
DBA_TAB_COLUMNS	OWNER	Owner of the object.
DBA_TAB_COLUMNS	DATA_DEFAULT	Default value for the column
DBA_TAB_COLUMNS	DEFAULT_LENGTH	Length of default value for the column
DBA_TAB_COLUMNS	COLUMN_ID	Sequence number of the column as created
DBA_TAB_COLUMNS	NULLABLE	Does column allow NULL values?
DBA_TAB_COLUMNS	DATA_SCALE	Digits to right of decimal point in a number
DBA_TAB_COLUMNS	DATA_PRECISION	Length: digits (NUMBER) or characters (CHAR,RAW)
DBA_TAB_COLUMNS	DATA_LENGTH	Length of the column in bytes
DBA_TAB_COLUMNS	DATA_TYPE	Datatype of the column
DBA_TAB_COLUMNS	COLUMN_NAME	Column name
DBA_TAB_COLUMNS	TABLE_NAME	Table, view or cluster name
DBA_TAB_COLUMNS	OWNER	Owner of the table, view or cluster
DBA_TAB_COMMENTS	COMMENTS	Comment on the object
DBA_TAB_COMMENTS	TABLE_TYPE	Type of the object
DBA_TAB_COMMENTS	TABLE_NAME	Name of the object
DBA_TAB_COMMENTS	OWNER	Owner of the object

DBA_TAB_GRANTS	CREATED	Timestamp for the grant
	INDEX_PRIV	Permission to create/drop an INDEX on the object?
	ALTER_PRIV	Permission to ALTER the object?
	REFERENCES_PRIV	Permission to make REFERENCES to the object?
	UPDATE_PRIV	Permission to UPDATE the object?
	DELETE_PRIV	Permission to DELETE from the object?
	INSERT_PRIV	Permission to INSERT into the object?
	SELECT_PRIV	Permission to SELECT from the object?
	GRANTOR	Name of the user who performed the grant
	TABLE_NAME	Name of the object
	OWNER	Owner of the object
	GRANTEE	User to whom access was granted
DBA_TS_QUOTAS	MAX_BLOCKS	User's quota in ORACLE blocks. NULL if no limit
	BLOCKS	Number of ORACLE blocks charged to the user
	MAX_BYTES	User's quota in bytes. NULL if no limit
	BYTES	Number of bytes charged to the user
	USERNAME	User with resource rights on the tablespace
	TABLESPACE_NAME	Tablespace name
DBA_USERS	EXPIRES	Password expiration date
	CREATED	User creation date
	TEMPORARY_TABLESPACE	Default tablespace for temporary tables
	DEFAULT_TABLESPACE	Default tablespace for data
	DBA_PRIV	Does the user have DBA privilege?
	RESOURCE_PRIV	Does the user have RESOURCE privilege?
	CONNECT_PRIV	Does the user have CONNECT privilege?
	PASSWORD	Encrypted password
	USER_ID	ID number of the user
	USERNAME	Name of the user

DBA_VIEWS	TEXT	View text
	TEXT_LENGTH	Length of the view text
	VIEW_NAME	Name of the view
DICTIONARY	Description of data dictionary tables and views	
	COMMENTS	Text comment on the object
	TABLE_NAME	Name of the object
DICT_COLUMNS	Description of columns in data dictionary tables and views	
	COMMENTS	Text comment on the object
	COLUMN_NAME	Name of the column
	TABLE_NAME	Name of the object that contains the column
TABLE_PRIVILEGES	Grants on objects for which the user is the grantor, grantee, or owner, or PUBLIC	
	CREATED	Timestamp for the grant
	INDEX_PRIV	Permission to create/drop an INDEX on the object
	ALTER_PRIV	Permission to ALTER the object?
	REFERENCES_PRIV	Permission to make REFERENCES to the object
	UPDATE_PRIV	Permission to UPDATE the object?
	DELETE_PRIV	Permission to DELETE from the object?
	INSERT_PRIV	Permission to INSERT into the object?
	SELECT_PRIV	Permission to SELECT from the object?
	GRANTOR	Name of the user who performed the grant
	TABLE_NAME	Name of the object
	OWNER	Owner of the object
	GRANTEE	Name of the user to whom access was granted
USER_AUDIT_CONNECT	Audit trail entries for user logons/logoffs	
	RETURNCODE	Oracle error code generated by the action. Zero if the action succeeded.
	SESSIONID	Numeric ID for each Oracle session.
	LOGOFF_DLOCK	Deadlocks detected during the session.
	LOGOFF_LWRITE	Logical writes for the session.
	LOGOFF_PREAD	Physical reads for the session.
	LOGOFF_LREAD	Logical reads for the session.

	LOGOFF_TIME	Timestamp for user logoff.
	ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION.
	TIMESTAMP	Timestamp for the user's logon.
	TERMINAL	Identifier for the user's terminal.
	USERHOST	Numeric instance ID for the Oracle instance from which the user is accessing the
	USERNAME	Name (not ID number) of the user whose actions were audited.
USER_AUDIT_RESOURCE		
	RETURNCODE	Oracle error code generated by the action. Zero if the action succeeded.
	STATEMENTID	Numeric ID for each statement run (a statement may cause many actions).
	ENTRYID	Numeric ID for each audit trail entry in the session.
	SESSIONID	Numeric ID for each Oracle session.
	ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION.
	OBJ_NAME	Name of the object affected by the action.
	OWNER	Intended creator of the non-existent object.
	TIMESTAMP	Timestamp for the creation of the audit trail entry.
	TERMINAL	Identifier for the user's terminal.
	USERHOST	Numeric instance ID for the Oracle instance from which the user is accessing the
	USERNAME	Name (not ID number) of the user whose actions were audited.
USER_AUDIT_TRAIL	Audit trail entries relevant to the user	
	RETURNCODE	Oracle error code generated by the action. Zero if the action succeeded.
	STATEMENTID	Numeric ID for each statement run (a statement may cause many actions).
	ENTRYID	Numeric ID for each audit trail entry in the session.
	SESSIONID	Numeric ID for each Oracle session.
	COMMENT_TEXT	Text comment on the audit trail entry.
	LOGOFF_DLOCK	Deadlocks detected during the session.
	LOGOFF_LWRITE	Logical writes for the session.

	LOGOFF_PREAD	Physical reads for the session.
	LOGOFF_LREAD	Logical reads for the session.
	LOGOFF_TIME	Timestamp for user logoff.
	SES_ACTIONS	Session summary. A string of 11 characters, one for each action type.
	GRANTEE	The name of the grantee specified in a GRANT/REVOKE statement.
	PRIVILEGE	Privileges granted/revoked by a GRANT/REVOKE statement.
	NEW_NAME	The new name of an object renamed by a RENAME statement.
	ACTION_NAME	Name of the action type corresponding to the numeric code in ACTION.
	ACTION	Numeric action type code. The corresponding name of the action type.
	OBJ_NAME	Name of the object affected by the action.
	OWNER	Creator of object affected by the action.
	TIMESTAMP	Timestamp for the creation of the audit trail entry.
	TERMINAL	Identifier for the user's terminal.
	USERHOST	Numeric instance ID for the Oracle instance from which the user is accessing the
	USERNAME	Name (not ID number) of the user whose actions were audited.
USER_CATALOG	Tables, views, synonyms, sequences accessible to the user	
	TABLE_TYPE	Type of the object
	TABLE_NAME	Name of the object
USER_CLUSTERS	Descriptions of user's own clusters	
	PCT_INCREASE	Percentage increase in extent size
	MAX_EXTENTS	Maximum number of extents allowed in the segment
	MIN_EXTENTS	Minimum number of extents allowed in the segment
	NEXT_EXTENT	Size of secondary extents in bytes
	INITIAL_EXTENT	Size of the initial extent in bytes
	MAX_TRANS	Maximum number of transactions
	INI_TRANS	Initial number of transactions
	KEY_SIZE	Estimated size of cluster key plus associated rows

	PCT_USED	Minimum percentage of used space in a block
	PCT_FREE	Minimum percentage of free space in a block
	TABLESPACE_NAME	Name of the tablespace containing the cluster
	CLUSTER_NAME	Name of the cluster
USER_CLU_COLUMNS	Mapping of table columns to cluster columns	
	TAB_COLUMN_NAME	Key column in the table
	TABLE_NAME	Clustered table name
	CLU_COLUMN_NAME	Key column in the cluster
	CLUSTER_NAME	Cluster name
USER_COL_COMMENTS	Comments on columns of user's tables and views	
	COMMENTS	Comment on the column
	COLUMN_NAME	Column name
	TABLE_NAME	Object name
USER_COL_GRANTS	Grants on columns for which the user is the owner, grantor, or grantee	
	CREATED	Timestamp for the grant
	REFERENCES_PRIV	Permission to make REFERENCES to the column?
	UPDATE_PRIV	Permission to UPDATE the column?
	GRANTOR	Name of the user who performed the grant
	COLUMN_NAME	Name of the column
	TABLE_NAME	Name of the object
	OWNER	Username of the owner of the object
	GRANTEE	Name of the user to whom access was granted
USER_COL_GRANTS_MADE	All grants on columns of objects owned by the user	
	CREATED	Timestamp for the grant
	REFERENCES_PRIV	Permission to make REFERENCES to the column?
	UPDATE_PRIV	Permission to UPDATE the column?
	GRANTOR	Name of the user who performed the grant
	COLUMN_NAME	Name of the column
	TABLE_NAME	Name of the object
	GRANTEE	Name of the user to whom access was granted

USER_COL_GRANTS_REC	Grants on columns for which the user is the grantee
CREATED	Timestamp for the grant
REFERENCES_PRIV	Permission to make REFERENCES to the column?
UPDATE_PRIV	Permission to UPDATE the column?
GRANTOR	Name of the user who performed the grant
COLUMN_NAME	Name of the column
TABLE_NAME	Name of the object
OWNER	Username of the owner of the object
USER_CROSS_REF	Cross references for user's views, synonyms, and constraints
REF_DB_LINK	Database link of the referenced object
REF_TABLE_NAME	Name of the referenced object
REF_OWNER	Owner of the referenced object
TABLE_TYPE	Type of the referencing object
TABLE_NAME	Name of the referencing object
USER_DB_LINKS	Database links owned by the user
CREATED	Creation time of the database link
HOST	SQL*Net string for connect
PASSWORD	Password for logon
USERNAME	Name of user to log on as
DB_LINK	Name of the database link
USER_EXTENTS	Extents comprising segments owned by the user
BLOCKS	Size of the extent in ORACLE blocks
BYTES	Size of the extent in bytes
EXTENT_ID	Extent number in the segment
TABLESPACE_NAME	Name of the tablespace containing the extent
SEGMENT_TYPE	Type of the segment
SEGMENT_NAME	Name of the segment associated with the extent
USER_FREE_SPACE	Free extents in tablespaces accessible to the user
BLOCKS	Size of the extent in ORACLE blocks
BYTES	Size of the extent in bytes
BLOCK_ID	Starting block number of the extent
FILE_ID	ID number of the file containing the extent

	TABLESPACE_NAME	Name of the tablespace containing the extent
USER_INDEXES	Description of the user's own indexes	
	PCT_INCREASE	Percentage increase in extent size
	MAX_EXTENTS	Maximum number of extents allowed in the segment
	MIN_EXTENTS	Minimum number of extents allowed in the segment
	NEXT_EXTENT	Size of secondary extents in bytes
	INITIAL_EXTENT	Size of the initial extent in bytes
	MAX_TRANS	Maximum number of transactions
	INI_TRANS	Initial number of transactions
	TABLESPACE_NAME	Name of the tablespace containing the index
	UNIQUENESS	Uniqueness status of the index: "UNIQUE" or "NONUNIQUE"
	TABLE_TYPE	Type of the indexed object
	TABLE_NAME	Name of the indexed object
	TABLE_OWNER	Owner of the indexed object
	INDEX_NAME	Name of the index
USER_IND_COLUMNS	Columns comprising user's indexes or on user's tables	
	COLUMN_LENGTH	Indexed length of the column
	COLUMN_POSITION	Position of column within index
	COLUMN_NAME	Column name
	TABLE_NAME	Table or cluster name
	INDEX_NAME	Index name
USER_OBJECTS	Objects owned by the user	
	MODIFIED	Timestamp for the last DDL change to the object
	CREATED	Timestamp for the creation of the object
	OBJECT_TYPE	Type of the object
	OBJECT_ID	Object number of the object
	OBJECT_NAME	Name of the object

USER_SEGMENTS	Storage allocation for all database segments	
	MAX_EXTENTS	Maximum number of extents allowed in the segment
	EXTENTS	Number of extents allocated to the segment
	BLOCKS	Size, in Oracle blocks, of the segment
	BYTES	Size, in bytes, of the segment
	TABLESPACE_NAME	Name of the tablespace containing the segment
	SEGMENT_TYPE	Type of segment: "TABLE", "CLUSTER", "INDEX", "ROLLBACK", "DEFERRED ROLLBACK",
	SEGMENT_NAME	Name, if any, of the segment
USER_SEQUENCES	Description of the user's own sequences	
	LAST_NUMBER	Last sequence number written to disk
	CACHE_SIZE	Number of sequence numbers to cache
	ORDER_FLAG	Are sequence numbers generated in order?
	CYCLE_FLAG	Does sequence wraparound on reaching limit?
	INCREMENT_BY	Value by which sequence is incremented
	MAX_VALUE	Maximum value of the sequence
	MIN_VALUE	Minimum value of the sequence
	SEQUENCE_NAME	SEQUENCE name
USER_SYNONYMS	The user's private synonyms	
	DB_LINK	Database link referenced in a remote synonym
	TABLE_NAME	Name of the object referenced by the synonym
	TABLE_OWNER	Owner of the object referenced by the synonym
	SYNONYM_NAME	Name of the synonym
USER_TABLES	Description of the user's own tables	
	BACKED_UP	Has table been backed up since last modification?
	PCT_INCREASE	Percentage increase in extent size
	MAX_EXTENTS	Maximum number of extents allowed in the segment
	MIN_EXTENTS	Minimum number of extents allowed in the segment

	NEXT_EXTENT	Size of secondary extents in bytes
	INITIAL_EXTENT	Size of the initial extent in bytes
	MAX_TRANS	Maximum number of transactions
	INI_TRANS	Initial number of transactions
	PCT_USED	Minimum percentage of used space in a block
	PCT_FREE	Minimum percentage of free space in a block
	CLUSTER_NAME	Name of the cluster, if any, to which the table belongs
	TABLESPACE_NAME	Name of the tablespace containing the tablespace
	TABLE_NAME	Name of the table
USER_TABLESPACES	Description of accessible tablespaces	
	STATUS	Tablespace status: "ONLINE" or "OFFLINE"
	PCT_INCREASE	Default percent increase for extent size
	MAX_EXTENTS	Default maximum number of extents
	MIN_EXTENTS	Default minimum number of extents
	NEXT_EXTENT	Default incremental extent size
	INITIAL_EXTENT	Default initial extent size
	TABLESPACE_NAME	Tablespace name
USER_TAB_AUDIT_OPTS	Auditing options for user's own tables and views	
	UPD	Auditing UPDATE WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	SEL	Auditing SELECT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	REN	Auditing RENAME WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	LOC	Auditing LOCK WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	INS	Auditing INSERT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	IND	Auditing INDEX WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	GRA	Auditing GRANT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	DEL	Auditing DELETE WHENEVER SUCCESSFUL / UNSUCCESSFUL.

	COM	Auditing COMMENT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	AUD	Auditing AUDIT WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	ALT	Auditing ALTER WHENEVER SUCCESSFUL / UNSUCCESSFUL.
	TABLE_TYPE	Type of the object: "TABLE" or "VIEW"
	TABLE_NAME	Name of the object
USER_TAB_COLUMNS		
	DATA_DEFAULT	Default value for the column
	DEFAULT_LENGTH	Length of default value for the column
	COLUMN_ID	Sequence number of the column as created
	NULLABLE	Does column allow NULL values?
	DATA_SCALE	Digits to right of decimal point in a number
	DATA_PRECISION	Length: digits (NUMBER) or characters (CHAR,RAW)
	DATA_LENGTH	Length of the column in bytes
	DATA_TYPE	Datatype of the column
	COLUMN_NAME	Column name
	TABLE_NAME	Table, view or cluster name
USER_TAB_COMMENTS		
	COMMENTS	Comment on the object
	TABLE_TYPE	Type of the object: "TABLE" or "VIEW"
	TABLE_NAME	Name of the object
USER_TAB_GRANTS		
	CREATED	Timestamp for the grant
	INDEX_PRIV	Permission to create/drop an INDEX on the object?
	ALTER_PRIV	Permission to ALTER the object?
	REFERENCES_PRIV	Permission to make REFERENCES to the object?
	UPDATE_PRIV	Permission to UPDATE the object?
	DELETE_PRIV	Permission to DELETE from the object?
	INSERT_PRIV	Permission to INSERT into the object?
	SELECT_PRIV	Permission to SELECT from the object?
	GRANTOR	Name of the user who performed the grant
	TABLE_NAME	Name of the object

OWNER	Owner of the object
GRANTEE	Name of the user to whom access was granted
USER_TAB_GRANTS_MADE All grants on objects owned by the user	
CREATED	Timestamp for the grant
INDEX_PRIV	Permission to CREATE/DROP INDEX on the object?
ALTER_PRIV	Permission to ALTER the object?
REFERENCES_PRIV	Permission to make REFERENCES to the object?
UPDATE_PRIV	Permission to UPDATE the object?
DELETE_PRIV	Permission to DELETE from the object?
INSERT_PRIV	Permission to INSERT into the object?
SELECT_PRIV	Permission to SELECT from the object?
GRANTOR	Name of the user who performed the grant
TABLE_NAME	Name of the object
GRANTEE	Name of the user to whom access was granted
USER_TAB_GRANTS_REC'D Grants on objects for which the user is the grantee	
CREATED	Timestamp for the grant
INDEX_PRIV	Permission to create/drop an INDEX on the object?
ALTER_PRIV	Permission to ALTER the object?
REFERENCES_PRIV	Permission to make REFERENCES to the object?
UPDATE_PRIV	Permission to UPDATE the object?
DELETE_PRIV	Permission to DELETE from the object?
INSERT_PRIV	Permission to INSERT into the object?
SELECT_PRIV	Permission to SELECT from the object?
GRANTOR	Name of the user who performed the grant
TABLE_NAME	Name of the object
OWNER	Owner of the object

USER_TS_QUOTAS	Tablespace quotas for the user
	MAX_BLOCKS User's quota in ORACLE blocks. NULL if no limit
	BLOCKS Number of ORACLE blocks charged to the user
	MAX_BYTES User's quota in bytes. NULL if no limit
	BYTES Number of bytes charged to the user
	TABLESPACE_NAME Tablespace name
USER_USERS	Information about the current user
	EXPIRES Password expiration date
	CREATED User creation date
	TEMPORARY_TABLESPACE Default tablespace for temporary tables
	DEFAULT_TABLESPACE Default tablespace for data
	DBA_PRIV Does user have DBA Privilege?
	RESOURCE_PRIV Does user have RESOURCE privilege?
	CONNECT_PRIV Does user have CONNECT privilege?
	USER_ID ID number of the user
	USERNAME Name of the user
USER_VIEWS	Text of views owned by the user
	TEXT View text
	TEXT_LENGTH Length of the view text
	VIEW_NAME Name of the view

The Dynamic Performance Tables

Version 6.0 of the ORACLE RDBMS contains a set of underlying "tables" that are maintained by the RDBMS and accessible to the DBA user SYS. These tables are called *dynamic performance tables* because they are continuously updated while a database is open and in use, and their contents relate primarily to performance.

Although these tables appear to be regular database tables, they are not. Like ROWIDs and ROWNUMS, these tables may be selected from, but never updated or altered.

Access to the Dynamic Performance Tables

After installation, only username SYS has access to the dynamic performance tables. However, access to these tables is required for any user needing to view the MONITOR displays available in SQL*DBA.

If a DBA wants to make information in these tables accessible to other database users, he can create views on individual tables and then grant users SELECT access to the views.

Alternatively, the file MONITOR.SQL may be run to grant access to PUBLIC on all of the dynamic performance tables. The drawback to running this file is that you may not want all database users to have equal access to all of this information. Thus, the selective method of creating views and granting access may be preferable. See the *Installation and User's Guide* for details on running MONITOR.SQL.

Tables Used by MONITOR Displays

If you want to give non-DBAs access to SQL*DBA so they can use the MONITOR screens to monitor realtime database use, you must also give these users access to the tables required by those displays. Thus, if any user other than SYS wants to use MONITOR, he must have been given access to one or more of the dynamic performance tables.

If you want to selectively grant access, use the following table to determine which tables are of interest.

<i>Monitor Option</i>	<i>Requires Table(s)</i>
FILE	V\$DBFILE V\$FILESTAT
IO	V\$STATNAME V\$PROCESS V\$SESSION V\$SESSTAT V\$SYSSTAT
LATCH	V\$LATCHNAME V\$LATCHHOLDER V\$LATCH
LOCK	V\$LOCK V\$PROCESS V\$RESOURCE
PROCESS	V\$PROCESS
ROLLBACK	V\$ROLLNAME V\$ROLLSTAT
STATISTICS	V\$STATNAME V\$PROCESS V\$SESSION V\$SYSTAT V\$SESSTAT
TABLE	V\$ACCESS
USER	V\$PROCESS V\$SESSION V\$BGPROCESS V\$LATCH V\$LATCHNAME V\$LOCK V\$RESOURCE V\$TRANSACTION

Description of Individual Tables These tables are identified by the prefix V\$; a list of the table and column names follows.

Table V\$ACCESS

DOS cui

PID
TABLE#
OWNER#
OWNERNAME
TABLENAME

Table V\$BGPROCESS

Nom

Describes the background processes.

PADDR
NAME
DESCRIPTION
ERROR

Table V\$DBFILE

uu

Information about each database file in the database.
FILE#
NAME

sys01
out01
rbs01
tmp01
dat01
ud01

Table V\$LATCH

owl

Information about each type of latch.

ADDR
LATCH#
LEVEL#
WAITS
IMMEDIATES
TIMEOUTS
NOWAITS
SUCCESSES

17

Table V\$LATCHHOLDER

✓

Information about the current latchholders.
PID
LADDR

LADDR
0210436E

Table V\$LATCHNAME

owl

The decoded latchnames for the latches shown in table V\$LATCH.
LATCH#
NAME

Table V\$LOCK

owl

LADDR
PADDR
RADDR
LMODE
REQUEST

Table V\$LOCK	ADDR PID TY ID1 ID2 LMODE REQUEST
Table V\$PROCESS	Information about currently active processes. ADDR PID SPID USERNAME TERMINAL PROGRAM → C:\ORACCEG\PBIN\SQLDBA.COM BACKGROUND LATCHWAIT LOCKWAIT
Table V\$RESOURCE	ADDR TY ID1 ID2
Table V\$SESSION	Session information for each current session. SID PADDR SESSION# USER# USERNAME COMMAND TADDR
Table V\$SESSTAT	For each current session, the current statistics values. SID STATISTIC VALUE
Table V\$STATNAME	Decoded statistic names for the statistics shown in the tables V\$SESSTAT and V\$SYSSTAT. <u>Statistic#</u> NAME CLASS

Table V\$SYSSTAT

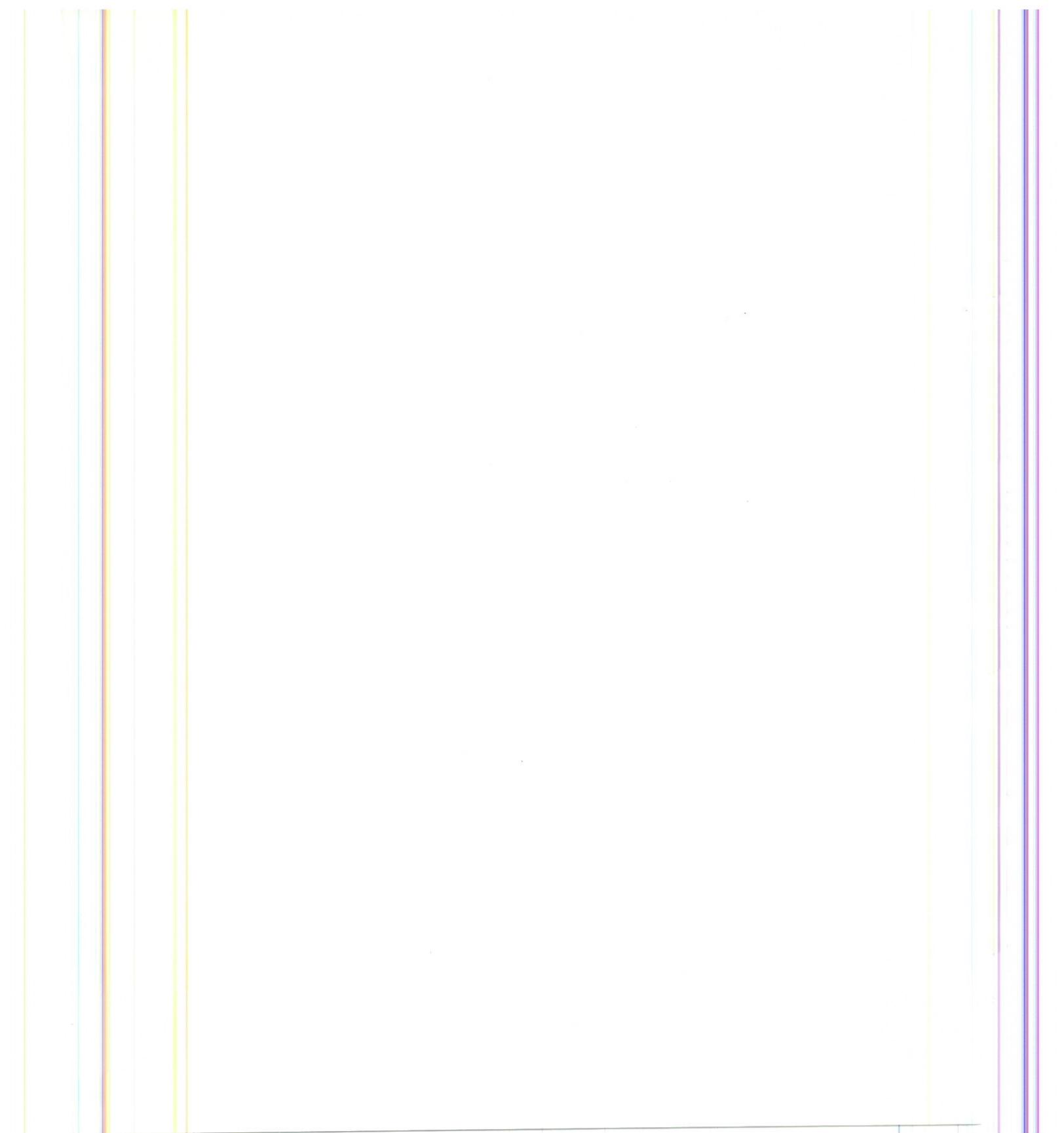
The current system-wide value for each statistic in table V\$SESSTAT.

<u>STATISTIC#</u>	<u>VALUE</u>
-------------------	--------------

Table V\$TRANSACTION

<i>com</i>	ADDR
	SCNBASE
	SCNWRAP
	XIDUSN
	XIDSLOT
	XIDSQN
	UBADBA
	UBASQN
	UBAREC

ADDR
SCNBASE
SCNWRAP
XIDUSN
XIDSLOT
XIDSQN
UBADBA
UBASQN
UBAREC



F

NATIONAL LANGUAGE SUPPORT

This appendix describes features introduced in Version 6 to enable ORACLE installations to operate in various languages, and to enable ORACLE applications to process multi-lingual data stored in ORACLE databases.

Topics include:

- description of National Language Support features
- specifying national language operation on an instance basis
- using National Language Support features.

This chapter contains material of interest to DBAs and application developers who must support users who wish to use national language features.

What does National Language Support Provide?

National Language Support fulfills two main requirements:

- full support within the RDBMS for handling data in multiple, national language character sets
- an end-user interface that enables the operation of end-user applications completely in the national language of the user.

Version 6 supports all single byte (7 and 8-bit) character sets used for European Latin languages (for example, English, French, German, Spanish) and enables end-user interfaces for these languages to be implemented.

Thus, the RDBMS and utilities support several features:

- the simultaneous use of multiple 7 and 8-bit character sets by one ORACLE instance
- sorting of character data according to language-specific alphabetic conventions
- ORACLE messages displayed to users in their local language
- month and day names spelled in local languages
- calculation of the week number according to the ISO standard.

Each supported NLS language requires an additional set of files for that language. Every ORACLE RDBMS is supplied with the one language native to the country. Additional sets of language files are available for separate purchase.

Specifying National Language Operation

Two INIT.ORA parameters control the operation of national language features: LANGUAGE and NLS_SORT. Refer to Appendix D for descriptions of these two parameters.

In summary, the LANGUAGE parameter specifies three arguments (*language*, *territory*, and *char_set*) in the form:

LANGUAGE = *language_territory.char_set*

The default value is:

AMERICAN_AMERICA.US7ASCII

The parameter NLS_SORT may have the values TRUE or FALSE; the default, FALSE, indicates sorting should be based on the numeric values of the characters regardless of which character set is specified.

To specify that an instance running on a DEC VAX/VMS system should operate in French, the following two values would be used:

```
LANGUAGE = French_France.we8dec  
NLS_SORT = true
```

The effects of these parameter values are illustrated in examples throughout this appendix.

Language Used for ORACLE Messages

Messages are always returned in the character set specified in the LANGUAGE parameter. Given the specification of the two non-default parameters above, since the *language* argument of the LANGUAGE parameter is "French", then all ORACLE messages are returned to the user application in French.

ORACLE messages are stored in external message files, by product, as in one file for RDBMS messages, another for SQL*Forms messages. Multiple message files exist, one for each supported language for each supported product. The filenames for the message files follow the format:

```
utility-language_id.MSB
```

For example, for the French language the RDBMS message file is named ORAF.MSB and for SQL*Forms (the IAP portion) it is named IAPF.MSB. For *utility* and *language_id*, standard ORACLE acronyms are used.

Message files are binary files that are generated from the original message text and cannot be edited directly.

For example, when "French" is specified, the message:

```
ORA-00942: table or view does not exist
```

appears as:

```
ORA-00942: table ou vue n'existe pas
```

Message files are stored in one particular character set (usually the main character set for the particular machine environment). If necessary, the message text is converted from this character set to the specified one, if they are different.

For example, the message:

```
ORA-01005: null password given: login denied
```

will appear in French as:

```
ORA-01005: aucun mot de passe; connexion refusée
```

since the LANGUAGE parameter specified the DEC 8-bit character set. If, however, the LANGUAGE parameter had been specified as:

```
LANGUAGE = French_France.US7ASCII
```

then the message would appear as:

```
ORA-01005: aucun mot de passe; connexion refusee
```

Since the 7-bit ASCII character set does not contain an acute *e* (é), it is replaced by an unaccented *e* before the message is displayed. If the character set used does have an é, then the message always appears as in the first example, no matter what the character set is. The conversion of text between character sets and the use of replacement characters follow the same technique as the CONVERT function, described later in this appendix.

Individual message files may impose different length restrictions on the messages; see the comments at the top of the message files.

Language Used for Day and Month Names

The value used for the *language* argument of the LANGUAGE parameter also determines the language, and therefore, spelling, of day and month names used by the functions TO_CHAR and TO_DATE. Thus, continuing with our French example, the query:

```
SELECT TO_CHAR(SYSDATE, 'Day:Dd Month yyyy')  
FROM DUAL
```

would return:

```
Mercredi :10 Février 1988
```

Month and day name abbreviations are also in the language specified, so the query could just as easily return:

```
Me :10 Fév 1988
```

Default date format (DD-MON-YY) uses the language-specific month name abbreviations. For example, the above date would be inserted using:

```
INSERT INTO tablename VALUES ('10-Fév-88')
```

Only day and month name spellings in the local language are supported. Numbers spelled using the TO_CHAR function always use English spellings, as in the following example:

```
SELECT TO_CHAR(TO_DATE('24-Fv-88'), 'Day: ddspth Month')
FROM DUAL
```

would return:

```
Mercredi : twenty-fourth Février
```

Week and Day Number Calculation

The *territory* argument of the LANGUAGE parameter determines the calculation used for week numbering and the convention used for day numbering. You have two choices for week number calculations:

- the ISO standard
- the calculation based on the number of days from Jan 1 (the default).

Using the ISO standard, week numbers range from 1 to either 52 or 53. Every week is assumed to start on a Monday and end on a Sunday. Whether a week belongs to one year or the next depends on the day on which Jan 1 falls:

- If Jan 1 falls on a Friday, Saturday, or Sunday, then the week including Jan 1 is the last week of the previous year, because most of the days in the week belong to the previous year.
- If Jan 1 falls on a Monday, Tuesday, Wednesday, or Thursday, then the week is the first week of the new year, because most of the days in the week belong to the new year.

For example, Jan 1, 1987 was a Thursday, so Monday Dec 28, 1987 to Sunday Jan 3, 1988 was week 53. Monday Jan 4, 1988 to Sunday Jan 10, 1988 was week 1.

(Note that the calculation for the *number of days* from Jan 1 always takes week 1 to be Jan 1 to Jan 7, week 2 to be Jan 8 to Jan 14, and so on. The day of the week is irrelevant.)

You also have two choices for day numbering:

- day 1 is a Sunday (the default)
- day 1 is a Monday.

Given a territory argument of "France", Monday will be used as day 1 and the ISO standard will be used for week numbering.

Processing Data in Multiple Character Sets

To understand the problems addressed by National Language Support, it is useful to first describe general features of character representation and character sets.

What are Character Sets?

The appearance of characters on a terminal depends on the convention for character representation used by the terminal. When you press a character key on the keyboard, the terminal generates a number according to the convention being used by the terminal. When the terminal receives as input a number representing a character, it displays the character shape according to the convention.

The convention is specified through a *character set definition*, which determines the numbers used to represent each character that the terminal is capable of handling.

Problems of Handling Multi-lingual Data

The lack of comprehensive standards for character representation causes various problems when terminals that use different character sets handle common data. For example, the widely used ASCII 7-bit standard character set defines the numbers used to represent the 26 characters in the English alphabet. However, other alphabets use many characters (such as é, à, ç, ü, å, ß) that are not included in the ASCII standard. In fact, no universal standard defines the numbers used to represent these characters, and in practice, each hardware manufacturer has defined its own standards.

Another problem is that there are more of these characters than can be included in a 7-bit character set or even an 8-bit character set. Thus, hardware manufacturers have generally defined several language-specific 7-bit standards, providing some of the characters used in the various languages, and 8-bit standards containing as many of the characters as can be included.

For example, there is a 7-bit DEC character set for French that includes the characters à, ç, é, ù, è, and û. This character set maintains the ASCII values for the English alphabet, and replaces the characters @,\,{,|,}, and ~ with the characters listed. There are also DEC 7-bit character sets for German, Spanish, and other languages. Several 8-bit standards have also been defined by, for example, DEC, HP, and IBM (for the PC), each of which allocates different numbers to the "European" characters.

The only agreement between the various 7 and 8-bit character sets is that they follow the ASCII standard for the numbers used for the 26 characters in the English alphabet. However, the various IBM EBCDIC

character set standards use a completely different numbering scheme. Also, the numbers used for punctuation characters, as in [] { } and |, are generally different between various character sets.

Storing Character Data

When a character string is entered into a database, what is actually stored are the numbers generated, according to the character set used by the terminal. If the character string is later retrieved from a terminal using a different character set, some characters may not display correctly.

Suppose two character sets use the same number, 97, for *a* (as in the ASCII standard). The *a* will display correctly when passed between the terminals. Suppose however, that the character set used to enter the data uses the value 123 for an *é* but the character set used to retrieve the data uses a value of 233. In this case, the *é* will be displayed incorrectly, as whatever character is defined as number 123, for example, a "[". This would actually be the case for data entered by a user from a terminal using the DEC 7-bit French character set and retrieved by a user from a terminal using the DEC 8-bit character set.

The solution to these convention disparities requires the ability to manipulate character data as entered in any possible character set, including the conversion of character data from one character set to another.

SQL's Approach to Different Character Sets

Converting Between Upper and Lower Case

The SQL functions UPPER, LOWER, and INITCAP must correctly handle uppercase and lowercase conversions. Because the numbers used to represent *à* and *À* vary according to the character set definitions, the functions must know which character set to use in order to convert case correctly.

The character set used is the one specified by the argument *char_set* in the parameter LANGUAGE. The functions convert case according to the conventions of the character set specified.

Converting Between Character Sets

A new SQL function, `CONVERT`, allows for the conversion between character sets. The full syntax for `CONVERT` follows (and is described in the *SQL Language Reference Manual*):

```
CONVERT(column[,dest_char_set[,source_char_set]])
```

where:

- | | |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>column</code> | is the database column, literal, or expression to be converted. The data will automatically be converted into a character string if required. |
| <code>dest_char_set</code> | is the name of the character set to which the data is to be converted. It can be a database column containing the name of the character set, or a literal (datatype CHAR). The default value is US7ASCII. |
| <code>source_char_set</code> | is the name of the character set in which the data is stored in the database. It can be a database column containing the name of the character set, or a literal (datatype CHAR). The default value is the current value of the <i>char_set</i> argument of the <code>LANGUAGE</code> parameter. |

In the following example, the `CONVERT` function is used to generate a report from data entered using the DEC 8-bit character set for output to a printer using the HP 8-bit character set:

```
SELECT CONVERT (ENAME, 'WE8HP','WE8DEC') FROM EMP
```

Using Replacement Characters

Problems may arise in both case and data conversion because character sets do not include all the characters that may be required.

A case conversion problem arises when the corresponding upper or lower case character is not included in the character set. For example, the 7-bit DEC French character set contains an à but not an Å, whereas the DEC 8-bit character set includes both. If the 7-bit French character set has been specified in the `LANGUAGE` parameter then what should `UPPER('à')` return? In these cases, a replacement character is chosen that provides a sensible alternative to the actual character required. In this example, an A would be returned. If no corresponding uppercase/lowercase value exists (and no replacement character is defined) then the same character is returned.

Replacement characters are also used, when required, for converting data between character sets. For example, suppose the character Å has been inserted into a column, COL1, of table T1 using a terminal operating with the DEC 8-bit character set. Suppose now that a user

wants to retrieve this data using a terminal operating in the DEC 7-bit French character set. What should the following return?

```
SELECT CONVERT(COL1, 'F7DEC', 'WE8DEC') FROM t1
```

The replacement character is used, if available, as for case conversion. Thus, an A would be returned.

When no replacement character is defined problems also arise. For example, suppose that COL1 contained the German lowercase sharp s, "ß". This character can be entered using the DEC 8-bit character set, but does not exist in the French 7-bit character set, and no sensible replacement character can be defined. In these cases a general default replacement character is used to indicate that a conversion cannot be made. In this example a "?" would be returned.

The REPLACE Function

An additional problem arises in the special case of "double" characters. An example is the sharp s "ß". The uppercase form of this character is "SS" (although the converse is not always the case). This double character is not provided as a single number value in the various character sets, and since only single byte conversions are supported, UPPER('ß') returns 'ß'.

A new function, REPLACE, helps overcome problems like this one. The syntax follows:

```
REPLACE(column[,src_string[,dest_string]])
```

where:

column is the database column or literal (datatype CHAR) in which a text string should be replaced.

src_string is the source string which should be replaced by the destination string. If the source string is empty, the string will be returned unchanged.

dest_string is the destination string which should replace the source string. If the destination string is empty, the occurrences of the source string will be removed.

If both source and destination strings are empty, the function returns NULL.

The sharp s can then be handled by using, for example:

```
SELECT REPLACE(UPPER('Stra e','  ','SS')) FROM DUAL
```

which returns STRASSE.

Sorting Character Data Typically when character data is sorted, the sort sequence is based purely on the numeric values of the characters. This is called a *binary sort* and produces the expected results for the English alphabet because the ASCII and EBCDIC standards define the letters *a* to *z* in ascending numeric value. Note however, that in the ASCII standard all uppercase letters appear before any lowercase letters. In the EBCDIC standard, the opposite is true: all lowercase letters appear before any uppercase letters.

When data contains the additional European characters, a binary sort does not produce the expected results. For example, the character strings ABC, ABZ, BCD, ÅBC would be returned by an ORDER BY query in the sequence shown if a binary sort is used. To request ordering with European characters in mind, you can request that a sort be performed based on the character set and sort sequence expected. Thus the above strings would be returned in the desired order: ABC, ÅBC, ABZ, BCD.

Using Non-Default Sorting To request other than the standard binary sort, enter the value TRUE for the INIT.ORA parameter NLS_SORT (for which FALSE is the default).

When NLS_SORT is set to TRUE, then the sequence used for sorting depends on the character set. For example, the US7ASCII character set definition specifies the binary sort method. If the *char_set* argument of the LANGUAGE parameter specifies US7ASCII, then an ORDER BY query will automatically use a binary sort, even if NLS_SORT is set to TRUE. The DEC 7-bit French character set definition, however, specifies an NLS sort. Hence, if the *char_set* argument specifies F7DEC, then an ORDER BY query will use the NLS sort.

The specification of national language sorting sequences is complicated because several different conventions may be in use for a particular language. For example, three different sorting sequences are used in Germany. This problem can be solved by providing multiple (but uniquely named) versions of each character set, one for each sorting sequence desired. Also, other versions of the standard ASCII and EBCDIC character sets can be defined so that upper and lower case letters are sorted together.

Another problem for sorting is presented by the "double" characters. For example, the Spanish double character *ll* is treated as a single character for sorting, and appears between *l* and *m* as illustrated by the sequence lz, lla, ma. However, sorting of double characters in this way is not supported and they are treated as separate characters in both the NLS and binary sorts.

Comparing Character Strings

When character strings are compared in a WHERE clause, the comparison is always made using the numeric values of the characters. For example, suppose COL1 contains the values ABC, ABZ, BCD, and ÅBC. When using the DEC 8-bit character set, the following query:

```
SELECT COL1 FROM T1 WHERE COL1 > 'B%'
```

will return both BCD and ÅBC. This is because Å has a higher numeric value in the character set than B, and the string comparison is based purely on the numeric values.

Specifying the LANGUAGE Parameter on a User Basis

To this point, the discussion of the LANGUAGE parameter has applied to all users of one instance accessing a database. All Oracle utilities also use the LANGUAGE parameter by default.

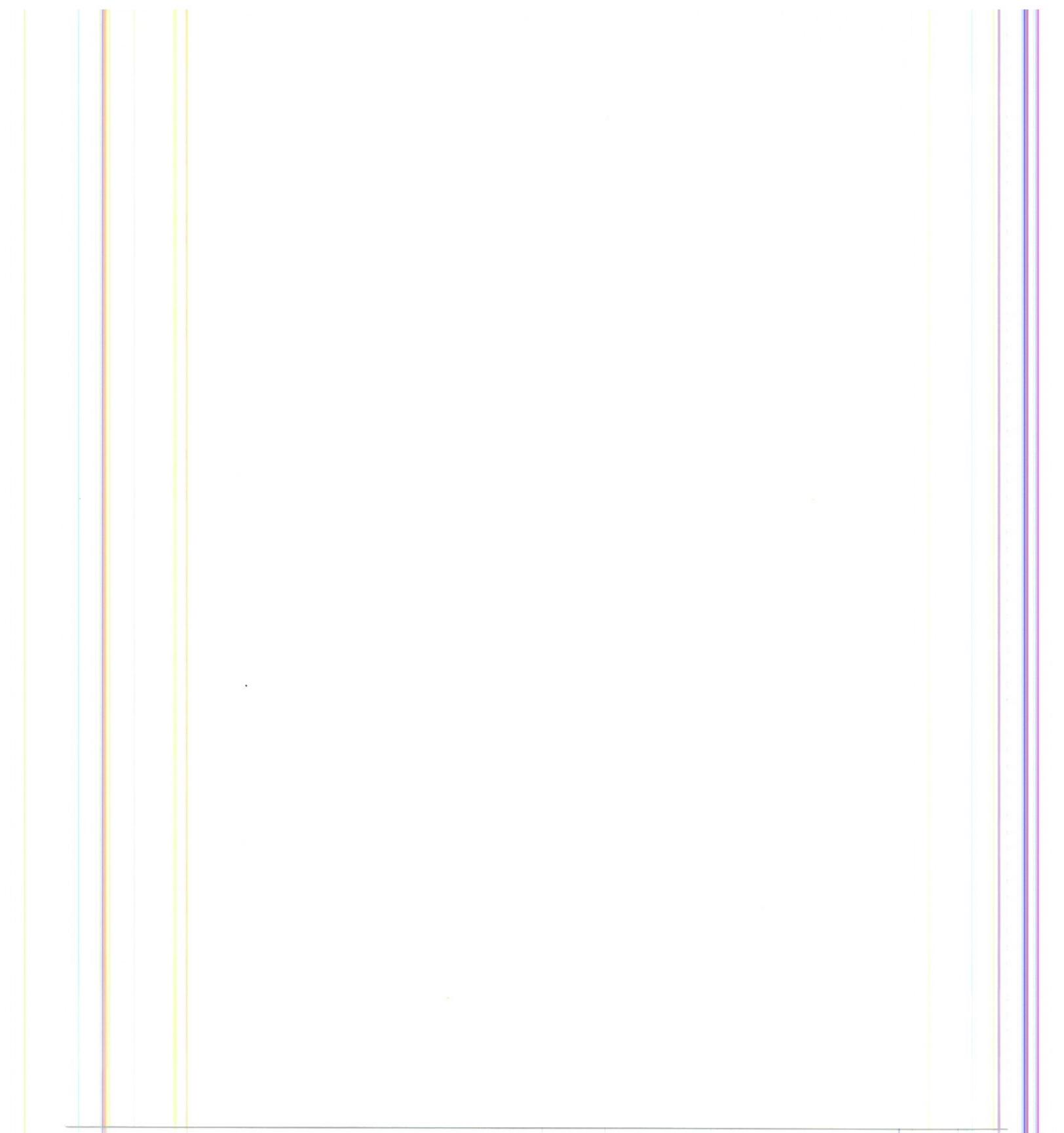
The LANGUAGE parameter, however, can be specified as a user-specific environment variable, to override certain effects of the instance-wide LANGUAGE parameter. In this case, the value set for LANGUAGE affects only utility-specific messages. The value of the default instance LANGUAGE parameter will still control system-wide:

- kernel RDBMS error messages
- language used for month and day names
- behavior of UPPER, LOWER, INITCAP, and CONVERT
- the sort method used for an ORDER BY query.

A user-specific LANGUAGE parameter enables users to simultaneously execute applications that have been especially designed to operate in different languages on the same ORACLE instance.

For example, a SQL*Forms application could be used in a multi-lingual environment to access data from a single database, if the application designer provided multiple versions of the application, each with the application messages in one of the desired languages.

The user-environment LANGUAGE parameter could then be set to the language used by each user. Each user then sees application and SQL*Forms messages in the language and character set chosen. Recall that any kernel messages the user sees are returned in the language and character set specified by the INIT.ORA LANGUAGE parameter, and that the behavior of the features mentioned earlier is controlled by this parameter, not the user environment LANGUAGE parameter.



G

SQL STATEMENT REFERENCE

This appendix is an alphabetical reference of the SQL statements used primarily by DBAs. This material is also found in the *SQL Language Reference Manual*, which contains reference material for all SQL statements supported by the ORACLE RDBMS. You can find the following SQL statements in this appendix:

- ALTER DATABASE
- ALTER ROLLBACK SEGMENT
- ALTER TABLESPACE
- ALTER USER
- AUDIT (all forms)
- CREATE DATABASE
- CREATE ROLLBACK SEGMENT
- CREATE TABLESPACE
- DROP ROLLBACK SEGMENT
- DROP TABLESPACE
- GRANT (all forms)
- NOAUDIT (all forms)
- REVOKE (all forms)

ALTER DATABASE

Purpose Alter an existing database in several ways:

- mount a database, either shared or exclusive
- open or close a database
- add or drop a redo log file
- rename a redo log file or a database file
- specify that the redo log will be archived and thus useful for media recovery, or will not be archived and thus useful only for instance recovery.

Prerequisites Requires DBA privilege.

Syntax

```
ALTER DATABASE [ database ]
[ADD LOGFILE filespec [REUSE] [, filespec [REUSE]] ...]
[DROP LOGFILE 'filename' [, 'filename'] ...]
[RENAME FILE 'filename' [, 'filename'] ...]
TO filename [,filename] ...
[ARCHIVELOG | NOARCHIVELOG]
[MOUNT [SHARED | EXCLUSIVE] | DISMOUNT]
[OPEN | CLOSE [NORMAL | IMMEDIATE]]
```

Keywords and Parameters	<i>database</i>	is an identifier for the database which cannot exceed 8 characters in length. If omitted, the value for the parameter DB_NAME in the current INIT.ORA file is assumed. This identifier is not related to the SQL*Net database specification.
	<i>ADD LOGFILE filespec</i>	<i>filespec</i> is a specification of a database file in the form: 'filename' [SIZE integer [K M]] <i>integer</i> specifies a SIZE in a number of bytes. No <i>integer</i> means ORACLE will use the default logfile size of 500K. If the <i>integer</i> is followed by K, then SIZE is computed as the <i>integer</i> multiplied by 1024. If M is used, then SIZE is <i>integer</i> multiplied by 1048576.

DROP LOGFILE <i>filename</i>	is the name of a current redo log file to be dropped.
RENAME FILE <i>filename</i> TO <i>filename</i>	specifies old and new file names for database files. New filenames must conform to your standard operating system file naming conventions.
ARCHIVELOG NOARCHIVELOG	enables and disables archiving, respectively.
MOUNT DISMOUNT	indicates whether to change the status of the database to MOUNTed or to DISMOUNTed. A database must be both MOUNTed and OPEN for most activities. Some DBA maintenance functions require the database to be MOUNTed but not OPEN.
OPEN CLOSE	indicates whether the database should be OPEN and available for normal access or CLOSED and unavailable for normal use.

Usage notes

Valid options depend on the current state of the database.

If the database is:

You may use:

not mounted

MOUNT only

mounted but not open

all arguments except MOUNT and CLOSE

open

CLOSE only

SIZE and REUSE

If SIZE is specified and REUSE is omitted, the file is to be created with the specified size and must not already exist. If both SIZE and REUSE are specified, the file should be created if it does not exist, but if it exists the size is to be checked. The use of REUSE alone or no options indicates that the file must already exist and should be used as is.

Some arguments may require additional tasks of the DBA, such as allocating and preparing files for the REUSE option, or taking a backup, if enabling ARCHIVELOG mode.

Generally, it is recommended that the database be both closed and dismounted when ALTERing the DATABASE.

Examples

To mount a database exclusively:

```
ALTER DATABASE dbname MOUNT EXCLUSIVE
```

To open the currently mounted database:

```
ALTER DATABASE OPEN
```

To switch modes such that you will archive redo logs:

```
ALTER DATABASE ARCHIVELOG
```

To perform several file operations:

```
ALTER DATABASE
ADD LOGFILE dblog3.log SIZE 50000
DROP LOGFILE dblog2..log
RENAME FILE data5.db testdata.db
```

ALTER ROLLBACK SEGMENT

Purpose Alter an existing rollback segment by:

- making it PUBLIC
- altering the storage parameters.

Prerequisite Requires DBA privilege.

Syntax ALTER [PUBLIC] ROLLBACK SEGMENT *segment*
STORAGE *storage*

Keywords and Parameters	PUBLIC	If specified, indicates that the rollback segment should become a public one, available for use by any instance. A PUBLIC rollback segment may be made PRIVATE by dropping it and recreating it.
	<i>segment</i>	specifies the name of an existing rollback segment.
	STORAGE <i>storage</i>	see syntax description for the STORAGE clause.

Usage Notes The STORAGE clause of the ALTER ROLLBACK SEGMENT statement affects future space allocation in the specified rollback segment. Therefore, INITIAL and MINEXTENTS cannot be used to alter an existing rollback segment.

Examples To make the current rollback segment named RSPUBLIC a public segment:

```
ALTER PUBLIC ROLLBACK SEGMENT RSPUBLIC
```

To change the storage parameters for a segment named RSONE:

```
ALTER ROLLBACK SEGMENT RSONE
STORAGE (
NEXT 1000
MAXEXTENTS 20
PCTINCREASE 0
)
```

ALTER TABLESPACE

Purpose Alter an existing tablespace by:

- adding or renaming database file(s)
- changing default table space storage parameters
- taking the tablespace online or offline
- starting or stopping backup.

Prerequisites Requires DBA privilege.

Syntax

```
ALTER TABLESPACE tablespace
  [ADD DATAFILE filespec [REUSE] [,filespec [REUSE] ] ...]
  [RENAME DATAFILE 'filename' [,'filename'] ...
   TO 'filename' [,'filename'] ...]
  [DEFAULT STORAGE storage]
  [ONLINE | OFFLINE [ NORMAL | IMMEDIATE ]]
  [BEGIN BACKUP | END BACKUP]
```

Keywords and Parameters

ADD DATAFILE
filespec [REUSE]

tablespace is the name of an existing tablespace.

makes additional space available to the tablespace. You may add a file to a tablespace while it is online or offline. *filespec* is a specification of a database file in the form:

'*filename*' [SIZE *integer* [K|M]]

integer specifies a SIZE in a number of bytes. No *integer* means ORACLE will use the default tablespace size of 10M. If the *integer* is followed by K, then SIZE is computed as the *integer* multiplied by 1024. If M is used, then SIZE will be the *integer* multiplied by 1048576.

If SIZE is specified and REUSE is omitted, the file is to be created with the specified size and must not already exist. If both SIZE and REUSE are specified, the file should be created if it does not exist, but if it exists the size is to be checked. The use of REUSE alone or no options indicates that the file must already exist and should be used as is.

RENAME
DATAFILE *filename*
TO *filename*

renames one or more files associated with the tablespace. Does not rename the operating system file associated with the tablespace file. The tablespace should be offline during the rename.

DEFAULT STORAGE <i>storage</i>	specifies the new default storage for future objects created in the tablespace. See the syntax description for the STORAGE clause.
ONLINE	signifies that the tablespace should be brought back online.
OFFLINE	signifies that the tablespace should be taken offline, either IMMEDIATEly or after all current users are no longer accessing the named tablespace (NORMAL).
BEGIN BACKUP	signifies that the database files that comprise this tablespace will be backed up with a system backup procedure. This statement has no effect on access to tablespace; users can continue to access it. This statement is necessary for control file and redo log record keeping.
END BACKUP	signifies that the system backup procedure has completed.

- Usage Notes** Rename a data file(s) with the following steps:
1. Take the tablespace offline.
 2. Rename the file(s) at the operating system level.
 3. Rename the database file(s) with the ALTER TABLESPACE statement.
 4. Bring the tablespace back online.
- Examples** To signal to the database that a backup is about to begin, enter:
`ALTER TABLESPACE ACCOUNTING BEGIN BACKUP`
- To signal to the database that the backup is finished, enter:
`ALTER TABLESPACE ACCOUNTING END BACKUP`
- Note that you may also backup a tablespace while it is online. An offline backup does not require that you archive redo logs.

ALTER USER

Purpose Alter any of the following characteristics of an existing database user:

- password
- default tablespace for object creation
- default temporary tablespace for temporary segments created on behalf of the user.

Prerequisites Requires DBA privilege.

Syntax

```
ALTER USER username [IDENTIFIED BY password]
[DEFAULT TABLESPACE tablespace]
[TTEMPORARY TABLESPACE tablespace]
```

Keywords and Parameters

username is the currently valid username for a database user.

password is the new password for the specified username.

DEFAULT TABLESPACE specifies the default tablespace for object creation.

TEMPORARY TABLESPACE specifies the default tablespace for the creation of temporary segments.

Usage Notes A user's password may also be changed with the GRANT statement. A user's default tablespace may also be set when a user is granted resource to a tablespace and the tablespace is the first tablespace to which the user has been granted resource. The default temporary tablespace will also be set to the first tablespace granted to a user.

Example To change username SCOTT's password to LION and change his default tablespace for the creation of database objects to an existing tablespace named TTEST, enter:

```
ALTER USER SCOTT
IDENTIFIED BY LION
DEFAULT TABLESPACE TTEST
```

AUDIT (Form I)

Purpose Select auditing options for database access.

Prerequisites Requires DBA privilege.

Syntax AUDIT {*system_option* [, *system_option*]... | ALL }
[WHENEVER [NOT] SUCCESSFUL]

Keywords and Parameters *system_option* is one of DBA, CONNECT, RESOURCE, or NOT EXISTS.

ALL is equivalent to a list of all the *system_options*.

By default, statements are audited whether or not they are successful.

WHENEVER SUCCESSFUL indicates that SQL statements are audited only if they complete successfully.

WHENEVER NOT SUCCESSFUL indicates that SQL statements are audited only when they fail.

Usage notes This form of auditing allows the DBA to track actions resulting from:

- connection to or disconnection from the database (CONNECT system option)
- statements requiring RESOURCE privilege (RESOURCE system option) such as the creation of database objects
- statements requiring DBA privilege (DBA system option)
- statements that return ORACLE error ORA-942, "table or view does not exist" (NOT EXISTS system option).

Auditing is enabled system-wide with the INIT.ORA parameter AUDIT_TRAIL. The AUDIT SQL statements execute successfully whether or not auditing is enabled. If auditing is enabled, rows are written to the audit trail. If it is not, no rows are written.

All forms of auditing place the audit results into data dictionary tables. As a result, you may easily generate reports based on the audit data dictionary tables. As the audit data dictionary tables grow, you may also archive old audit records (and then delete these same records). You should never drop the audit data dictionary tables themselves.

Note that for DDL and DML statements, the term "completion" means the completion of the operation's "parse phase" (parse, dictionary lookup, security check), which is not precisely the same as its

"execution". Note also that a rollback will not delete the row in the audit trail.

If neither WHENEVER SUCCESSFUL nor WHENEVER NOT SUCCESSFUL is specified, then the SQL statements are audited regardless of success or failure.

Examples To audit all successful resource requests, enter:

```
AUDIT RESOURCE WHENEVER SUCCESSFUL
```

To audit all unsuccessful attempts to access the database, enter:

```
AUDIT CONNECT WHENEVER NOT SUCCESSFUL
```

AUDIT (Form II)

Purpose	Enable auditing of access to database objects such as tables and views.
Prerequisites	Requires that you own the object to be audited or requires DBA privilege. The DEFAULT option requires DBA privilege.
Syntax	<pre>AUDIT {table_option [,table_option]... ALL} ON { [user.]table DEFAULT} [BY {SESSION ACCESS}] [WHENEVER [NOT] SUCCESSFUL]</pre>
Keywords and Parameters	<p><i>table_option</i> For tables, the applicable options are ALTER, AUDIT, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT, or UPDATE. For views, the applicable options are AUDIT, COMMENT, INSERT, DELETE, GRANT, LOCK, RENAME, SELECT, and UPDATE. For sequences, the applicable options are ALTER, AUDIT, GRANT, and SELECT. For synonyms, the applicable options are the same set of options for tables (since all the actions actually affect a base table).</p> <p><i>ALL</i> can be used to indicate all applicable options.</p> <p><i>user</i> is the username of the table owner.</p> <p><i>table</i> is the name of a view, table, sequence, or a synonym denoting a view or table.</p> <p><i>DEFAULT</i> specifies that all tables created subsequently will be audited as specified by the table_option list (sets default system-wide auditing options).</p> <p><i>BY SESSION</i> specifies that only one audit record per type of audit access be generated for the duration of the user's session.</p> <p><i>BY ACCESS</i> specifies that one audit record per access per type of audit access be generated.</p>
Usage Notes	<p>The BY clause determines the granularity of the audit by specifying how often rows are written to AUDIT_TRAIL. BY SESSION will record that a certain type of access was made to the database object. BY ACCESS will record that the access was made and as well as how often.</p> <p>Some events happen much more frequently than others (for example, DML statements normally occur more often than GRANTS). The more</p>

detailed the auditing, the more audit records generated, so you should consider which statements are most important to audit.

In BY SESSION mode, a table has at most one session record in AUDIT_TRAIL per user session. Selected SQL statements are recorded by *updating* the row in AUDIT_TRAIL. The rows record how many times the selected statements executed successfully or unsuccessfully. No BY clause is equivalent to BY SESSION.

In BY ACCESS mode, selected SQL statements are recorded by *inserting* rows in AUDIT_TRAIL. For a DML operation, rows are written or updated at the completion of the operation's "parse phase."

Example To audit all sessions in which unsuccessful attempts occur to access the EMP table, enter:

```
AUDIT ALL ON EMP WHENEVER NOT SUCCESSFUL
```

To audit every individual attempted INSERT or UPDATE on the EMP table, enter:

```
AUDIT INSERT, UPDATE ON EMP BY ACCESS
```

CREATE DATABASE

Purpose Create a database, with options to:

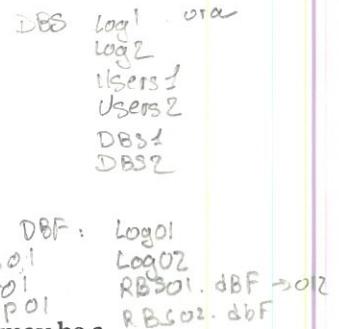
- set a maximum number of instances, database files, or redo log files
- specify names and sizes of database, and redo log file(s) to be used
- choose a mode of use for the redo log (ARCHIVELOG or NOARCHIVELOG)
- leave the resulting open database available in either shared or exclusive mode.

Prerequisites Requires DBA privilege.

Warning: this command prepares a database for initial use, and *erases all data currently in the files*.

Syntax

```
CREATE DATABASE [database]
  [CONTROLFILE REUSE]
  [LOGFILE filespec [,filespec] ...]
  [MAXLOGFILES integer ]
  [DATAFILE filespec [,filespec] ...]
  [MAXDATAFILES integer]
  [MAXINSTANCES integer]
  [ARCHIVELOG | NOARCHIVELOG]
  [SHARED | EXCLUSIVE]
```



Keyword and Parameters

database is the name of the database and may be a maximum of eight characters. The default database name is specified by the INIT.ORA parameter, DB_NAME. The database name is checked by the control file whenever the database is mounted or opened.

filespec is a specification of a database file in the form:

```
'filename' [SIZE integer [K|M]] [REUSE]
```

integer

specifies a SIZE in a number of bytes. No *integer* means ORACLE will use the default size of 10M for database files or the default size of 500K for logfiles. If the *integer* is followed by K, then SIZE is computed as the *integer* multiplied by 1024. If M is used, then SIZE will be the *integer* multiplied by 1048576.

REUSE

specifies that existing files should be reused, thus destroying any information they contain.

CONTROLFILE REUSE	specifies that existing control files identified by the INIT.ORA parameter, CONTROL_FILES, should be reused, thus ignoring and overwriting any information they may currently contain. Not usually used if this is the first time a database is created.
LOGFILE <i>filespec</i>	specifies one or more files to be used as redo log files. If none are specified, two default redo log files are created with names that are operating system dependent and default sizes.
MAXLOGFILES	specifies the maximum number of redo log files that can ever be created for this database. This number sets the absolute limit, and takes precedence over the INIT.ORA parameter LOG_FILES. This number cannot be increased except by re-creating a database. The valid range is 2 - 256. Overhead incurred by entering a higher number is negligible, affecting only the size of the control file.
DATAFILE <i>filespec</i>	specifies one or more files to be used for database files. If none are specified, one file is created with a name which is operating system dependent. All files specified for this argument become part of the SYSTEM tablespace.
MAXDATAFILES	specifies the maximum number of database files that can ever be created for this database. This number sets the absolute limit, and takes precedence over the INIT.ORA parameter DB_FILES. This number cannot be increased except by re-creating a database. The valid range is operating system dependent, typically 1 - 255. Overhead incurred by entering a higher number is negligible, affecting only the size of the control file. The default is 32.
MAXINSTANCES	specifies the maximum number of instances which can ever simultaneously mount and open this database. This number sets the absolute limit, and takes precedence over the INIT.ORA parameter INSTANCES. The valid range is 1 - 255.
ARCHIVELOG NOARCHIVELOG	specifies the initial mode of using redo log files. The mode can be changed via the SQL statement ALTER DATABASE. The default, NOARCHIVELOG, indicates that a redo log file can be reused without archiving its contents; thus the redo log will be used only for instance recovery but not for

media recovery. ARCHIVELOG indicates that redo log files will be archived before reuse, thereby preparing for the possibility of media recovery as well as instance recovery.

**SHARED
EXCLUSIVE**

specifies how the database should be left mounted after creation is complete. Applies only to this startup; after database creation this choice is made via the SQL*DBA command STARTUP statement by each instance opening a database. SHARED indicates that multiple instances may access this database. EXCLUSIVE indicates that only one instance may access it.

Usage Notes The CREATE DATABASE statement will erase all data in the specified database files in order to prepare them for initial database use. Thus, if the statement is used on an existing database, all data will be lost.

Examples To create a small database using defaults for all arguments:

```
CREATE DATABASE
```

To fully specify each argument:

```
CREATE DATABASE NEWTEST
  CONTROLFILE REUSE
  LOGFILE LOGONE.LOG 50000, LOGTWO.LOG 50000
  MAXLOGFILES 5
  DATAFILE 'DBONE.DAT' 100000
  MAXDATAFILES 10
  MAXINSTANCES 2
  ARCHIVELOG
  SHARED
```

CREATE ROLLBACK SEGMENT

Purpose Create a rollback segment, optionally specifying:

- whether the rollback segment may be allocated to any instance requesting it (PUBLIC) or only to instances naming it in their INIT.ORA files (private)
- the tablespace to contain the segment
- storage parameters.

Prerequisites Requires DBA privilege and RESOURCE privilege on the specified tablespace.

Syntax

```
CREATE [ PUBLIC ] ROLLBACK SEGMENT rollback_segment
      [ TABLESPACE tablespace ]
      [ STORAGE storage ]
```

Keyword and Parameters	PUBLIC	if specified, indicates that the rollback segment should be a public one, available for use by any instance. If omitted, indicates that the rollback segment should not be public.
	<i>rollback_segment</i>	is a valid ORACLE identifier, of 30 characters or less.
	TABLESPACE <i>tablespace</i>	names the tablespace in which the rollback segment will be created. The default tablespace is SYSTEM.
	STORAGE <i>storage</i>	specifies the storage allocation for the rollback segment. See the STORAGE clause for how to specify <i>storage</i> .

Usage Notes The tablespace must be online when a rollback segment is added to it. A tablespace may have multiple rollback segments. Generally, multiple rollback segments will improve performance.

Examples To create a rollback segment in the SYSTEM tablespace, enter:

```
CREATE PUBLIC ROLLBACK SEGMENT RBS_2
TABLESPACE SYSTEM
STORAGE (
    INITIAL 50000
    INCREMENT 50000
    MAXEXTENTS 10
)
```

CREATE TABLESPACE

Purpose Create a tablespace, specifying:

- the names of the files that comprise the tablespace
- default storage parameters
- whether the tablespace should be online or offline after creation.

Prerequisites Requires DBA privilege.

Syntax

```
CREATE TABLESPACE tablespace
    DATAFILE {filespec [, filespec] ...}
        [DEFAULT STORAGE storage]
        [ONLINE | OFFLINE]
```

Keywords and Parameters *tablespae* is an ORACLE identifier of up to 30 characters used to name the tablespace.

filespec is a specification of a database file in the form:
filename' [SIZE *integer* [K|M]] [REUSE]

REUSE specifies that any existing file(s) should be reused, thereby destroying any information the file(s) contain.

integer specifies a SIZE in a number of bytes. No SIZE means ORACLE will use the default size of 10M for tablespaces. If the *integer* is followed by K, then SIZE is computed as the *integer* multiplied by 1024. If M is used, then SIZE will be the *integer* multiplied by 1048576.

DATAFILE specifies the file or files that comprise the tablespace. The exact filename specification is operating system specific.

DEFAULT STORAGE are the default storage parameters for all objects created in the tablespace. The default tablespace storage is operating system specific. See the syntax description for the STORAGE clause for a complete description of storage specification.

ONLINE, OFFLINE makes the tablespace accessible or inaccessible respectively. The default, ONLINE, means that the tablespace will be available immediately for users who have been granted access to the tablespace. The data dictionary view DBA_TABLESPACES indicates the ONLINE or OFFLINE status of each tablespace.

Usage Notes A tablespace is the unit of backup and recovery within the database. It contains table, index, rollback, and temporary segments. All databases have at least one tablespace, SYSTEM, which is created when the database is created. Operating system files may be added to any tablespace dynamically while the tablespace is online or offline. Tablespaces may be dropped when the tablespace is online or offline. It is recommended that you take the tablespace offline before dropping it. By taking a tablespace offline, you insure that no transactions are open against the tablespace.

Before you can create a second tablespace, you must create a second rollback segment. You need not create a new rollback segment for each subsequent tablespace after the second.

Example To create a tablespace with one physical datafile, enter:

```
CREATE TABLESPACE TABSPACE_2
DATAFILE 'TABSPACE_FILE2.DAT' SIZE 20M
DEFAULT STORAGE (
  INITIAL 10K NEXT 50K
  MINEXTENTS 1 MAXEXTENTS 999
  PCTINCREASE 10
)
ONLINE
```

DROP ROLLBACK SEGMENT

Purpose Remove the specified rollback segment.

Prerequisites Requires DBA privilege.

Syntax `DROP [PUBLIC] ROLLBACK SEGMENT segment`

Keywords and Parameters `PUBLIC` must be specified for PUBLIC rollback segments.

`segment` specifies the name of an existing rollback segment.

Usage Notes Only rollback segments which are not in use can be dropped. Query the STATUS column of the DBA_ROLLBACK_SEGS data dictionary table to determine which rollback segments are in use. If the rollback segment is in use, you may either wait until all outstanding transactions using the rollback segment are complete, or you may SHUT the database down IMMEDIATE and then start it up in EXCLUSIVE mode.

All space allocated to the rollback segment is returned to the tablespace.

Examples To drop the public ACCOUNTING rollback segment, enter:

```
DROP PUBLIC ROLLBACK SEGMENT ACCOUNTING
```

DROP TABLESPACE

Purpose	Remove the specified tablespace, optionally removing all the tablespace's database objects, such as tables.
Prerequisites	Requires DBA privilege. The tablespace may be online or offline.
Syntax	<code>DROP TABLESPACE <i>tablespace</i> [INCLUDING CONTENTS]</code>
Keywords and Parameters	
<i>tablespace</i>	specifies the name of an existing tablespace. The SYSTEM tablespace may not be dropped.
INCLUDING CONTENTS	specifies that the tablespace is to be dropped even if it contains data. If the argument is omitted and the tablespace is empty, it is dropped. If the argument is omitted and the tablespace has contents, the tablespace is not dropped.
Usage Notes	You should take the tablespace offline before dropping it since the tablespace can not be dropped while users are accessing a tablespace's data, index, rollback or temporary segments.
Examples	To drop the MFRG tablespace and all its contents, enter: <code>DROP TABLESPACE MFRG INCLUDING CONTENTS</code>

GRANT (Form I)

Purpose	Provide access to the database.
Prerequisites	Requires DBA privilege, except any user may use the GRANT statement to change his or her own password.
Syntax	<pre>GRANT database_priv [,database_priv] ... TO user [,user] ... [IDENTIFIED BY password [,password] ...]</pre>
Keywords and Parameters	<p><i>database_priv</i> specifies one or more of the database privileges: DBA, CONNECT, RESOURCE.</p> <p><i>user</i> specifies either a new or existing database username. If the name already exists in the database, then this statement is used to add database privileges. If the name does not already exist, it is added to the data dictionary with the database privileges specified.</p> <p><i>password</i> specifies a password for each username specified. The list of usernames and passwords must be equal in number if multiple users are specified. A password may be used to provide new passwords to new users or to change the password of an existing user. A password need not be specified if the GRANT statement is being used to add privileges to an existing user.</p>
Usage Notes	<p>The CONNECT privilege establishes a new username in the database. With CONNECT privilege a user may connect to the database and operate on any objects to which he has been given privilege. For instance, all users may select from the data dictionary table DICTIONARY. A user may create views, synonyms and database links with the CONNECT privilege.</p> <p>The RESOURCE privilege permits a user to create database objects including tables, indexes, clusters, and sequences. This form of GRANT RESOURCE provides unlimited resource for all tablespaces. Use Form II to GRANT RESOURCE on specific tablespaces.</p> <p>The DBA privilege allows the user to bypass many standard privileges normally required to use database objects. The DBA privilege also permits the user to perform certain database administration tasks such as CREATE TABLESPACE and CREATE ROLLBACK SEGMENT.</p>

With the DBA privilege, a user may:

- SELECT from any table or view
- CREATE database objects for other users
- DROP other user's database objects including tables, views, synonyms, and database links
- GRANT various database privileges
- CREATE public synonyms and database links
- run full database exports and imports.

Examples To grant access and table-creation authority to a user named SCOTT, with the password TIGER, enter:

```
GRANT CONNECT, RESOURCE TO SCOTT IDENTIFIED BY TIGER
```

SCOTT may now create objects in the SYSTEM tablespace.

To grant the DBA privilege to an existing user named SCOTT, enter:

```
GRANT DBA TO SCOTT
```

A password need not be specified in this statement because SCOTT is an existing user.

SCOTT can change his password to COWSLIP by entering:

```
GRANT CONNECT TO SCOTT IDENTIFIED BY COWSLIP
```

GRANT (Form II)

Purpose	Provide access to database space (tablespaces) with the option of imposing a limit on the amount of space an individual user can use.
Prerequisite	Requires DBA privilege.
Syntax	<pre>GRANT RESOURCE [(quota [K M])] ON tablespace TO { PUBLIC user [,user]... }</pre>
Keywords and Parameters	<p>RESOURCE specifies that the list of users can create objects in the specified tablespace.</p> <p>quota is specified as an integer and represents the number of bytes of space within the tablespace that the user may allocate. No <i>quota</i> means an unlimited tablespace resource. The use of zero (0) revokes quota or RESOURCE privilege on the tablespace. If the <i>quota</i> is followed by K, then the quota is computed as the <i>quota</i> multiplied by 1024. If M is used, then the quota will be the <i>quota</i> multiplied by 1048576.</p> <p>tablespace specifies the name of an existing tablespace.</p> <p>user specifies an existing username. The keyword PUBLIC may be used to grant the resource privilege to all database users.</p>
Usage Notes	Form II of the GRANT statement provides additional flexibility and control as compared to Form I of the GRANT statement. The use of Form II is preferred so that DBAs can have additional control over how database space is allocated.
Example	If you are a DBA, you can grant SCOTT the ability to use a maximum of 10 megabytes of space in the FINANCE tablespace by entering: <pre>GRANT RESOURCE (10M) ON FINANCE TO SCOTT</pre>

GRANT (Form III)

Purpose	Provide various types of access to database objects, such as tables, views, and sequences.
Prerequisites	You must own the database object, have been granted GRANT OPTION on the object, or have the DBA privilege.
Syntax	<pre>GRANT {object_priv [,object_priv] ...} {ALL [PRIVILEGES]} ON [user.]object TO { user PUBLIC } [,user]... [WITH GRANT OPTION]</pre>
Keywords and Parameters	<p><i>object_priv</i> for tables, is one of: ALTER, DELETE, INDEX, INSERT, REFERENCES, SELECT or UPDATE. for views, is one of: DELETE, INSERT, SELECT, or UPDATE. for sequences, is either ALTER or SELECT. The UPDATE privilege can restrict updates to specific columns. The syntax for this privilege is: <code>GRANT UPDATE (column [,column]...) ...</code></p> <p><i>column</i> specifies a column from the <i>table</i>.</p> <p>ALL PRIVILEGES represents all of the privileges specified by <i>object_priv</i>.</p> <p>ON [user.] object specifies the table, view, or synonym on which the privileges are granted.</p> <p>TO user specifies the user(s) to which the privileges are granted.</p> <p>PUBLIC represents all users within the database and is used to grant privileges to all users, present and future.</p> <p>WITH GRANT OPTION specifies that the grantee may pass on the privileges that he has been granted to another user.</p>

Usage Notes	<p>The table owner always has all privileges on the table.</p> <p>ALTER is the privilege to use the ALTER TABLE statement.</p> <p>DELETE is the privilege to delete rows from the table.</p> <p>INDEX is the privilege to create an index on a table.</p> <p>INSERT is the privilege to insert rows into the table.</p> <p>REFERENCES is the privilege to refer to the table within a table or column constraint.</p> <p>SELECT is the privilege to query the table. To restrict SELECT on particular columns, create a view on the table with only those columns and grant SELECT on the view.</p> <p>UPDATE is the privilege to update rows in the table. If columns are specified, then only those columns may be updated.</p> <p>The privileges granted to specific users are independent of those granted to PUBLIC. For example, if Scott granted SELECT on the EMP table to Blake and PUBLIC and then Scott revoked SELECT on the EMP table from PUBLIC, Blake would still have access to EMP (even though Blake is a member of PUBLIC).</p> <p>You may limit access to a table by first creating a view on the base table and then by granting access to the view and not to the base table.</p> <p>GRANTS on synonyms are converted to GRANTS on the base table referenced by the synonym.</p>
Sequences	<p>To use a sequence, a user other than the owner must have been granted the SELECT privilege. The ALTER privilege is for the ALTER SEQUENCE statement. A user may pass his sequence privileges to another user if he has been granted the WITH GRANT OPTION privilege.</p> <p>ORACLE is able to determine that you are granting access to a sequence as opposed to a table because all objects (including sequences, tables, views, indexes, and clusters) owned by a user must have distinct names.</p>

Example To grant full access authority on the table BONUS to a user named JONES, including the ability to grant these privileges to other users, enter:

```
GRANT ALL ON BONUS TO JONES WITH GRANT OPTION
```

To authorize any user to query or update a table of golf handicaps, enter:

```
GRANT SELECT, UPDATE ON GOLF_HANDICAP TO PUBLIC
```

Sequence example To grant access on ELLY's ESEQ sequence to BLAKE, ELLY would enter:

```
GRANT SELECT ON ESEQ TO BLAKE
```

BLAKE could then generate a number from ELLY's sequence with the following statement:

```
SELECT ELLY.ESEQ.NEXTVAL FROM DUAL
```

NOAUDIT (Form I)

Purpose	Partially or completely reverse the effect of a prior system AUDIT statement. (See Form II of AUDIT)
Prerequisites	This form of NOAUDIT requires DBA privilege.
Syntax	<pre>NOAUDIT {system_option [,system_option]... ALL} [WHENEVER [NOT] SUCCESSFUL]</pre>
Keywords and Parameters	<p><i>system_option</i> is one of the following system options: CONNECT, DBA, NOT EXISTS, or RESOURCE.</p> <p>ALL represents all of the system options.</p> <p>WHENEVER [NOT] SUCCESSFUL specifies that auditing should be turned off for successful or unsuccessful system accesses. If no WHENEVER clause is used, auditing is turned off for both successful and unsuccessful.</p>
Usage Notes	See the AUDIT statement for a complete description of the system options.
Examples	To stop the auditing of successful resource requests, enter:

```
NOAUDIT RESOURCE WHENEVER SUCCESSFUL
```

NOAUDIT (Form II)

Purpose	Partially or completely reverse the effect of a prior AUDIT statement, or of the DEFAULT table auditing option.										
Prerequisites	You must own the object that NOAUDIT operates on. You must have DBA privilege to change the auditing options set for DEFAULT.										
Syntax	<pre>NOAUDIT {option [,option]... ALL} ON { object DEFAULT } [WHENEVER [NOT] SUCCESSFUL]</pre>										
Keywords and Parameters	<table><tr><td><i>option</i></td><td>For tables ALTER, AUDIT, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT, or UPDATE. For views AUDIT, DELETE, GRANT, INSERT, LOCK, RENAME, SELECT, or UPDATE. For sequences ALTER, AUDIT, GRANT, or SELECT. For synonyms, the applicable options are the same set of options for the table or view on which it is based.</td></tr><tr><td>ALL</td><td>represents all valid options.</td></tr><tr><td><i>object</i></td><td>is an existing table, view, or sequence you own.</td></tr><tr><td>DEFAULT</td><td>references the default auditing options set by the DBA to define auditing options for new tables. DEFAULT does not apply to views or sequences.</td></tr><tr><td>WHENEVER [NOT] SUCCESSFUL</td><td>specifies that auditing should be turned off for either successful or unsuccessful accesses to the <i>table</i>. If this clause is omitted, auditing is turned off for <i>both</i> successful and unsuccessful accesses.</td></tr></table>	<i>option</i>	For tables ALTER, AUDIT, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT, or UPDATE. For views AUDIT, DELETE, GRANT, INSERT, LOCK, RENAME, SELECT, or UPDATE. For sequences ALTER, AUDIT, GRANT, or SELECT. For synonyms, the applicable options are the same set of options for the table or view on which it is based.	ALL	represents all valid options.	<i>object</i>	is an existing table, view, or sequence you own.	DEFAULT	references the default auditing options set by the DBA to define auditing options for new tables. DEFAULT does not apply to views or sequences.	WHENEVER [NOT] SUCCESSFUL	specifies that auditing should be turned off for either successful or unsuccessful accesses to the <i>table</i> . If this clause is omitted, auditing is turned off for <i>both</i> successful and unsuccessful accesses.
<i>option</i>	For tables ALTER, AUDIT, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT, or UPDATE. For views AUDIT, DELETE, GRANT, INSERT, LOCK, RENAME, SELECT, or UPDATE. For sequences ALTER, AUDIT, GRANT, or SELECT. For synonyms, the applicable options are the same set of options for the table or view on which it is based.										
ALL	represents all valid options.										
<i>object</i>	is an existing table, view, or sequence you own.										
DEFAULT	references the default auditing options set by the DBA to define auditing options for new tables. DEFAULT does not apply to views or sequences.										
WHENEVER [NOT] SUCCESSFUL	specifies that auditing should be turned off for either successful or unsuccessful accesses to the <i>table</i> . If this clause is omitted, auditing is turned off for <i>both</i> successful and unsuccessful accesses.										
Usage Notes	NOAUDIT causes ORACLE to stop auditing the use of a table or view. As a DDL statement, the NOAUDIT statement also commits the current transaction.										
Examples	<p>To stop the auditing of attempted UPDATEs on the table EMP, enter:</p> <pre>NOAUDIT UPDATE ON EMP</pre> <p>To stop all auditing of sessions of unsuccessful uses of EMP, enter:</p> <pre>NOAUDIT ALL ON EMP WHENEVER NOT SUCCESSFUL</pre>										

REVOKE (Form I)

Purpose	Revoke database privileges from one or more users.
Prerequisites	Requires DBA privilege.
Syntax	<pre>REVOKE { [CONNECT] [,RESOURCE] [,DBA] } FROM user [,user] ...</pre>
Keywords and Parameters	<p>user is an existing user.</p> <p>CONNECT, RESOURCE, DBA is the privilege to be revoked from <i>user</i>. CONNECT, RESOURCE, and DBA have the same meanings as in GRANT.</p>
Usage Notes	<p>REVOKE returns success even when there are no privileges to revoke. REVOKE does not affect any objects owned by the user. For example, the REVOKE of CONNECT from a user named Blake would not affect Blake's SALES table. Other users who had previously been granted access to Blake's SALES table may continue to access the table. If Blake is regranted access to the database, he will once again own the SALES table. It is therefore recommended that you drop all objects owned by a user before revoking CONNECT from that user.</p>
Example	To revoke RESOURCE authority from users named SCOTT and JOLLY_ROGER enter:

```
REVOKE RESOURCE FROM SCOTT, JOLLY_ROGER
```

REVOKE (Form II)

Purpose Revoke space privilege from a specific tablespace.

Prerequisites Requires DBA privilege.

Syntax

```
REVOKE space_privilege ON tablespace  
FROM user [,user] ...
```

Keywords and Parameters

space_privilege is RESOURCE. Currently there is only one space privilege.

tablespace is an existing tablespace to which the user has been granted RESOURCE privileges.

user is an existing user who was previously granted RESOURCE privilege on *tablespace*.

Usage Notes This form denies the user the ability to create objects in the tablespace. No future objects may be created and current objects will not be permitted to allocate further database space.

Example To deny SCOTT the ability to create objects in the SYSTEM tablespace, enter:

```
REVOKE RESOURCE ON SYSTEM FROM SCOTT
```

The preceding statement is also equivalent to:

```
GRANT RESOURCE(0) ON SYSTEM TO SCOTT
```

REVOKE (Form III)

Purpose	Revoke access privileges from one or more users for tables, views, and sequences.										
Prerequisites	You must own the database object or have been granted WITH GRANT OPTION on the table.										
Syntax	<pre>REVOKE { object_priv [,object_priv]... ALL } ON [user.]object FROM { user PUBLIC } [,user]...</pre>										
Keywords and Parameters	<table><tr><td><i>object_priv</i></td><td>For tables, is one of: ALTER, DELETE, INDEX, INSERT, REFERENCES, SELECT or UPDATE. For views, is one of: DELETE, INSERT, SELECT, or UPDATE. For sequences, is either ALTER or SELECT.</td></tr><tr><td>ALL</td><td>revokes all privileges.</td></tr><tr><td>ON <i>object</i></td><td>specifies the name of the table, view, or sequence you are revoking privileges from. The table is a table you own or a table to which you have been granted WITH GRANT OPTION.</td></tr><tr><td><i>user</i></td><td>is the username of a user whose privileges are to be revoked. FROM PUBLIC revokes privileges originally granted to PUBLIC, but does not revoke individual grants on that same table.</td></tr><tr><td>PUBLIC</td><td>represents all users, present and future.</td></tr></table>	<i>object_priv</i>	For tables, is one of: ALTER, DELETE, INDEX, INSERT, REFERENCES, SELECT or UPDATE. For views, is one of: DELETE, INSERT, SELECT, or UPDATE. For sequences, is either ALTER or SELECT.	ALL	revokes all privileges.	ON <i>object</i>	specifies the name of the table, view, or sequence you are revoking privileges from. The table is a table you own or a table to which you have been granted WITH GRANT OPTION.	<i>user</i>	is the username of a user whose privileges are to be revoked. FROM PUBLIC revokes privileges originally granted to PUBLIC, but does not revoke individual grants on that same table.	PUBLIC	represents all users, present and future.
<i>object_priv</i>	For tables, is one of: ALTER, DELETE, INDEX, INSERT, REFERENCES, SELECT or UPDATE. For views, is one of: DELETE, INSERT, SELECT, or UPDATE. For sequences, is either ALTER or SELECT.										
ALL	revokes all privileges.										
ON <i>object</i>	specifies the name of the table, view, or sequence you are revoking privileges from. The table is a table you own or a table to which you have been granted WITH GRANT OPTION.										
<i>user</i>	is the username of a user whose privileges are to be revoked. FROM PUBLIC revokes privileges originally granted to PUBLIC, but does not revoke individual grants on that same table.										
PUBLIC	represents all users, present and future.										
Usage Notes	<p>This form revokes privileges on tables or views from one or more users. If a user has been authorized to access a table by more than one person, that user may continue to access the table until privileges have been revoked by all who granted it.</p> <p>Revoking a privilege from a user also revokes it from others to whom that user has granted it.</p>										

Example To revoke user JONES's privileges to make changes to the table DEPT10, enter:

```
REVOKE ALTER, DELETE, INSERT, UPDATE  
ON      DEPT10  
FROM    JONES
```

To revoke all privileges on DEPT10 from all users who do not have explicitly granted privileges, enter:

```
REVOKE ALL ON DEPT10 FROM PUBLIC
```

To revoke Blake's privileges on the ESEQ sequence, Elly would enter:

```
REVOKE ALL ON ESEQ FROM BLAKE
```

If Elly has DBA privilege, then Elly could revoke SELECT on Blake's BSEQ sequence from Scott with the following command:

```
REVOKE SELECT ON BLAKE.BSEQ FROM SCOTT
```

Storage clause

Purpose Specify characteristics of database storage for various database objects including tables, indexes, and clusters. Several SQL statements contain a phrase to define storage, usually prefixed with the keyword STORAGE or DEFAULT STORAGE. These statements include:

CREATE TABLESPACE
CREATE TABLE
CREATE CLUSTER
CREATE INDEX
CREATE ROLLBACK SEGMENT

as well as the statements to ALTER these objects.

Prerequisites none

Syntax

```
STORAGE (
    [ INITIAL integer ] [ NEXT integer ]
    [ MINEXTENTS integer ] [ MAXEXTENTS integer ]
    [ PCTINCREASE integer ]
)
```

Keywords and Parameters	<i>integer</i>	see the syntax description for <integer>.
	INITIAL	The size in bytes of the first extent allocated when the object is created — the original amount of space allocated to the object. The default initial extent size is 10240 bytes. The minimum INITIAL is 4096 bytes, the maximum is 4095 megabytes. All values are rounded to the nearest 1024 bytes.
	NEXT	The size in bytes of every subsequent extent to be allocated. Default is 10240 bytes. The minimum NEXT is 2048, the maximum is 4095 megabytes. All values are rounded to the nearest 1024 bytes. This size is a base value, which may remain constant for each new extent, or may change, depending on the value for PCTINCREASE.
	PCTINCREASE	If PCTINCREASE is zero (0) then every subsequent extent is the same size, the size specified by NEXT. If PCTINCREASE is positive (as in 10) then every subsequent extent will grow that percent (10%) over the previous extent allocated.
	MAXEXTENTS	The total number of extents, including the first, which can ever be allocated. Default is 9999 extents.

MINEXTENTS	The total number of extents to be allocated when the segment is created. This allows for a large allocation of space when an object is created, even if the space available is not contiguous. Default is 1 extent, meaning that just the initial extent is allocated.
PCTINCREASE	If MINEXTENTS is greater than 1, then the specified number of subsequent extents are allocated at CREATE time using the values INITIAL, NEXT, and PCTINCREASE. The percent by which each NEXT extent will grow over the last extent allocated. If PCTINCREASE is zero, then the size of each additional extent remains constant. The default PCTINCREASE is 50. The default of 50 allows the size of each extent to grow as the table grows. This requires fewer extents for a given table size, thereby reducing I/O. Each time NEXT is calculated, it will grow by PCTINCREASE. The result of PCTINCREASE multiplied by the size of the last extent is rounded up to the next multiple of a block size. PCTINCREASE cannot be negative.

Usage Notes The preceding storage parameters may affect both the time it takes to access data stored within the database as well as how efficiently space is used within the database.

When used with ALTER statements as in ALTER TABLE, the new storage allocations only affect future extents.

Example

```
STORAGE ( INITIAL 100K NEXT 50K
          MINEXTENTS 1 MAXEXTENTS 50
          PCTINCREASE 5
      )
```

H

DATABASE LIMITS

<i>Item</i>	<i>Limit</i>
blocks (ORACLE) / initial extent	2 blocks minimum (automatically enforced)
characters / CHAR column	255 characters maximum
characters / index	no absolute limit, but a function of block size
characters / LONG column	65,535 characters maximum
columns / index (or cluster index)	16 columns maximum
columns / table	254 columns maximum
columns or expressions / list	254 columns maximum
columns / view	254 columns maximum
columns (LONG) / table	1 LONG column
context area size	no absolute limit (1024 is the minimum initial extent size)
control files / database	one minimum: strongly recommend 2 or more on separate devices.
control file size	size varies, but is always small (usually less than 1K)

database file size minimum	no absolute limit except for first file whose minimum size is 500 Kbytes
database file size maximum	O/S dependent, typically 16 million ORACLE blocks
database files / system	255 or value of MAXDATAFILES in CREATE DATABASE . Ultimately, an operating system limit.
indexes / table	no limit
instances / shared disk system	O/S dependent, subject to ORACLE limit of 255
locks / transaction	unlimited
MAXEXTENTS	O/S dependent
nested queries	255 queries
NUMBER (max value)	$9.99\dots \times 10^{124}$
precision	up to 38 significant digits per numeric value
redo log files / database	255 or value for LOG_FILES in INIT.ORA, or by MAXLOGFILES in CREATE DATABASE. Ultimately, an operating system limit.
redo log file size	minimum 50 Kbytes
rollback segments / database	no limit
rows / table	no limit
tablespaces / database	no limit
tables / cluster	32
tables / database	no limit

GLOSSARY

access (database) One of three types of privileges to access and use the database. The DBA can grant a user one or more of the database privileges CONNECT, RESOURCE, or DBA.

access (database object) One of several types of privileges to use or alter database objects such as tables, views, or indexes. Database users can grant other users access such as SELECT, ALTER, or DROP, via the GRANT statement.

access path The path taken by the RDBMS to fetch the rows required to perform the SQL statement. Alternate paths are judged based on the efficiency of locating the data, using the known information about the data (provided by the WHERE clauses), the indexes existing on the data, and in the last case, full table scans.

address (row) See ROWID.

archive In a general sense, to save data for possible later use. In a specific sense, to save the data found in the online redo logs, in the even that the logs are needed to restore media. This kind of archiving is done using the SQL*DBA command ARCHIVE.

ARCH process One of five background processes used by multi-process database systems, the Archive process performs automatic archival of redo log files when the log is used in ARCHIVELOG mode.

array processing Processing performed on batches of data rather than one row at a time. In some ORACLE utilities such as Export/Import and the precompilers, users can set the size of the array; increasing the array size will generally improve performance.

auditing ORACLE installation and data dictionary features which allow the DBA and users to track usage of the database. Auditing information is stored in the data dictionary and SQL statements are used to control which information is stored. The DBA can set default auditing activity.

audit trail A database table written to by the RDBMS when auditing is enabled (by the INIT.ORA parameter) and auditing options have been selected by users or the DBA. One base table owned by SYS contains all auditing trail rows.

B*-tree A high performance indexing structure used by ORACLE to create and store indexes.

background process One of the five processes used by an instance of multiple-process ORACLE, to perform and coordinate tasks on behalf of concurrent users of a database. The six processes are named ARCH, DBWR, LGWR, PMON, SMON and they exist as long as an instance does.

base table A database table that does not depend on any other table. Contrast with *view*.

bind phase The phase of SQL statement processing during which all variables are made known to the RDBMS so that actual values to be used during execution of the statement.

bind variable A variable in a SQL statement which must be replaced with a valid value or address of a value in order for the statement to successfully execute.

block Basic unit of storage (physical and logical) for all ORACLE data. The number of blocks allocated per ORACLE table depends on the tablespace in which the table is created. The ORACLE block size varies by operating system and may differ from the block size of the host operating system

bootstrap segment A segment necessary during CREATE DATABASE to create the initial database structures (data dictionary).

buffers (database) Temporary storage place for database blocks which are currently being accessed and changed by database users.

buffers (redo log) Temporary storage place for redo log blocks which are currently being accessed and changed by database users.

caches Temporary holding places for database data that is currently being accessed or changed by users, or for data that the RDBMS requires to support users.

cache manager Responsible for making sure all changes made by software are propagated to disk in the right order.

chained row A row that is stored in more than one database block, and that therefore has several row pieces. Long rows (or LONG data) whose data is greater than the size of a block always have multiple row pieces.

CHAR datatype An ORACLE datatype which stores character strings to a maximum size of 255 characters.

checkpoint A point at which, on an instance basis, changed blocks are written to the database. Occurs when the number of redo log file blocks written equals LOG_CHECKPOINT_INTERVAL and also when an online redo log file fills (at log switch).

cleanup pass A pass made through the buffers in the SGA by background process PMON in order to clean up resources held by any processes that terminated abnormally.

client A general term for a user, software application, or computer which requires the services, data, or processing of another application or computer.

closed database A database which is associated with an instance (the database is mounted) but not open. Databases must be closed for a few database maintenance functions. Can be accomplished via the SQL statement ALTER DATABASE.

cluster A means of storing data from multiple tables together, when the data in those tables contains common information and is likely to be accessed concurrently.

cluster index An index manually created after a cluster has been created and before any DML statements can operate on the cluster. This index is created on the cluster key columns with the SQL statement CREATE INDEX.

cluster key The columns which clustered tables have in common, and which are chosen as the storage/access key.

cold start See *database creation*.

commit To make changes to data (inserts, updates, deletes) permanent. Before changes are stored, both the old and new data exist so that changes can be stored or the data can be restored to its prior state. When a user enters the SQL statement COMMIT, all changes from that transaction are made permanent.

communications protocol Any one of a number of standard means of connecting two computers together so that they can share information. Protocols consists of several layers of both software and hardware, and may connect homogeneous or heterogeneous computers.

composite keys A key value which is formed by more than one column.

concatenated index (or key) An index which is created on more than one column of a table. Can be used to guarantee that those columns are unique for every row in the table and to speed access to rows via those columns.

concurrency General term meaning the access of the same data by multiple users. In database software concurrency requires complex software programming to assure that all users see correct data and that all changes are made in the proper order.

connecting to ORACLE The act of providing the right combination of *username* and *password*, in order to identify yourself to ORACLE as a valid user (one who has been given privileges to access and use the database).

consistency (data) General database term and issue requiring that all related data be updated together in the proper order, and that if there is redundant data, all data is consistent.

context area A work area where ORACLE stores the current SQL statement, and if the statement is a query, the result's column headings and one row of the result. Synonym for *cursor*.

control file A small administrative file required by every database, necessary to start and run a database system. A control file is paired with a database, not with an instance. Multiple identical control files are preferred to a single file, for reasons of data security.

creating a database The process of making a database ready for initial use. Includes clearing the database files and loading initial database tables required by the RDBMS. Accomplished via the SQL statement CREATE DATABASE.

CRT An ORACLE RDBMS utility used to define or alter how terminals interact with ORACLE software. Most often used to define which keys perform which functions.

CRT file A file containing information about one hardware terminal, its keys and their mappings to ORACLE programs, and general escape sequences to enable different settings on the terminal.

cursor (1) Synonym for context area. Cursors may be *open* or *closed*.

cursor (2) A marker such as a blinking square or line, which marks your current position on a CRT screen.

cursor coordinates The location of a cursor on a CRT.

cursor descriptor An area reserved for every cursor that contains certain cursor information.

cursor ID The name given to each cursor.

database A set of dictionary tables and user tables which are treated as a unit.

database file A file used in a database. A database is made up of one or more tablespaces which in turn are made up of one or more database files.

database link A name given to a combination of information (such as database name, node, and protocol) that allows a user of one database to access data in another database.

database name A unique identifier used to name a database. Assigned in the CREATE DATABASE statement or in the INIT.ORA file.

database system A combination of an instance and a database. If the instance is started and connected to an open database, then the database system is available for access by users.

data control (DCL) statements A category of SQL statements which control access to the data and to the database.

data definition (DDL) statements A category of SQL statements which define or delete database objects such as tables or views.

data dictionary A set of tables and views owned by the DBA user SYS which is a central source of information for the ORACLE RDBMS itself and for users. The tables are installed when the database is created and are automatically maintained by ORACLE.

data locks Locks acquired either explicitly or implicitly by users on data. Data locks can be acquired in any of the six lock modes and appear as TX or TM locks in the MONITOR display.

data manipulation (DML) statements A category of SQL statements which query and update the actual data.

DATE A standard ORACLE datatype to store date and time data. Standard date format is 01-JAN-88.

DBA privilege Database privileges given via the GRANT DBA statement. Should be limited to very few users.

DBWR process One of five background processes used by multi-process database systems, the Database Writer process writes new data to the database

DDL lock One of two types of *dictionary locks*, acquired on behalf of users who are executing DDL statements. DDL locks are exclusive. Contrast to *parse locks*, the other type of dictionary lock

deadlock A rarely occurring situation where two or more user processes of a database cannot complete their transactions. This occurs because each process is holding a resource that the other process requires (such as a row in a table) in order to complete. Though these situations occur rarely, the RDBMS detects and resolves deadlocks by rolling back work of one of the processes.

declarative SQL statement A SQL statement which does not generate a call to the database and is therefore not an executable SQL statement. An example is BEGIN DECLARE SECTION or DECLARE CURSOR. Compare to *executable SQL statements*. Declarative statements are used primarily in precompiled and PL/SQL programs.

deferred rollback segment A rollback segment containing entries which could not be applied to the tablespace, because the given tablespace was offline. As soon as the tablespace comes back online, all the entries are applied.

define phase One phase of executing a SQL query, in which the program defines buffers to hold the results of a query to be executed.

dependent parameters INIT.ORA parameters which should not be altered by users or DBAs because the RDBMS automatically calculates their values based on the values of one or more other INIT.ORA parameters.

describe phase One phase of executing a SQL query, in which the program gathers information about the results of the query to be executed.

detached process See *background process*.

dictionary cache Any one of several caches of data dictionary information contained in the SGA. Caching dictionary information improves performance because the dictionary information is used frequently.

dictionary cache locks One of three types of internal locks, on entries in the dictionary caches.

dictionary locks Locks acquired on behalf of users who are parsing statements (shared dictionary locks or parse locks) or who are executing DDL statements (exclusive dictionary locks or DDL locks).

dismounted database A database which is not mounted by any instance, and thus cannot be opened and is not available for use.

distributed database A collection of databases that can be operated and managed separately and also share information.

distributed processing Performing computation (like PL/SQL programs) on multiple CPUs to achieve a single result.

distributed query A query which selects data from databases residing at multiple nodes of a network.

DML lock Synonym for *data locks*.

enqueue The lock on a given resource. Waiters for a given resource have not yet gotten the enqueue. Enqueues exist for many database resources.

EXCLUSIVE mode A type of lock on a resource that excludes any other access to that resource; the holder of the lock has exclusive rights to alter that resource.

execute phase One phase of SQL statement execution, when all the information necessary for execution is obtained and the statement is executed.

executable SQL statement A SQL statement which generates a call to the database. Includes almost all queries, DML, DDL, and DCL statements. Compare to *declarative SQL statements*.

Export An ORACLE utility used to write operating system files containing database data, which can later be restored via Import.

expression A portion of a SQL statement less than a complete statement. May refer to a WHERE clause, a FROM clause, or a criterion, using a function or mathematical function, to be applied to the data.

fetch One phase of query execution, where actual rows of data meeting all search criteria are retrieved from the database.

foreign key A value or column in one table that refers to a key in another table.

file The basic unit of information maintained by an operating system.

file management locks One of three types of internal locks, these locks exist on physical files (control files, redo log files, and database files) to make sure they are read and updated properly.

fragmented database A database which has been used for a while so that data belonging to various tables is spread all over the database and free space may be in many small pieces rather than fewer large pieces, due to much database activity or use. Fragmentation may make space usage less efficient and can be

remedied by exporting and importing some or all data.

free extents Extents of database blocks that have not yet been allocated to any table or index segment. Another term for free space.

full table scan A method of data retrieval in which the RDBMS directly searches in a sequential manner all the database blocks for a table (rather than using an index), when looking for the specified data.

grant option The option to pass on to other database users privileges which you have been granted. This option is granted using GRANT ... WITH GRANT OPTION.

Import An ORACLE utility used to read operating system files containing database data and written by Export, in order to restore the data to an ORACLE database.

index An optional structure associated with a table which is used by the RDBMS to locate rows of that table quickly, and optionally to guarantee that every row is unique. Database users create and drop indexes with SQL statements.

INIT.ORA file A file containing a list of parameters that is read when an instance is started, to configure the instance. The parameters help identify the database and also affect the size of the SGA, and therefore the performance of the instance.

instance A running ORACLE. There is always a one to one correspondence between an ORACLE instance and a system global area (SGA). An instance usually consists of an SGA, several background processes, and zero or more user processes.

an ORACLE instance and a system global area (SGA). An instance usually consists of an SGA, several background processes, and zero or more user processes.

instance identifier A means of distinguishing one instance from another when multiple instances exist on one CPU. The identifier is specified in an operating system dependent manner.

instance recovery Recovery in the event of software or hardware failure. This would occur if the software aborted abnormally, due to a severe bug, deadlock, or destructive interaction between programs. See also *media recovery*.

integrity (data) General database term meaning the accuracy, consistency and security of the data, as maintained by the RDBMS throughout all user access.

INTERNAL A username that the DBA can use to connect to a database in rare situations, such as database creation. Requires special operating system account privileges.

internal locks Locks on internal database structures and physical files that guarantee the integrity of the data. There are three types of internal locks: dictionary cache locks, file and log management locks, and tablespace and rollback segment locks.

Julian date An algorithm for expressing dates in integer form, using the JDATE function and date formatting. Julian dates allow additional arithmetic functions to be performed on dates.

kernel The core software that maintains an ORACLE RDBMS.

key Information used to identify rows in a table. Typically indexes are created on key columns.

latch An internal lock held for a short time to ensure the consistency of internal shared data structures.

LGWR process One of six background processes used by multi-process database systems, the Log Writer process writes redo log entries to disk.

lock A user level lock that is held for the duration of a transaction to ensure inter-transaction consistency.

lock mode One of six modes that a lock can be acquired in. The modes are SHARE, SHARE UPDATE, SHARE EXCLUSIVE, ROW SHARE, ROW EXCLUSIVE, EXCLUSIVE. Not all locks can be acquired in all modes; data locks may be.

log allocation The allocation of another part of an online log file to an instance for writing log entries. Relevant primarily for shared disk systems, when multiple instances all write to the same log file and are each given allocations in the current log file.

logical unit of work See *transaction*.

log sequence number A unique number identifying a specific online log file. Assigned by the RDBMS and used by the DBA during archiving and recovery of log files.

LONG datatype An ORACLE datatype which stores character strings to a maximum length of 65536 characters.

than ORACLE when multiple users update the same data and all updates are not made in the proper order. This never happens in ORACLE.

maintenance release The second identifying number of ORACLE software. In V6.0.10, the maintenance release number is 0.

media recovery Recovery in the event of hardware failure. Hardware failure would prevent reading or writing of data and thus operation of the database. See also *instance recovery*.

mounted database A database associated with an ORACLE instance. The database may be opened or closed. When a database is mounted, it has been verified to be "well-formed" and has a logical name by which subsequent statements may refer to it. A database must be both mounted and opened to be accessed by users. A database which has been mounted but not opened can be accessed by DBAs for some maintenance purposes.

multiple-process A mode of database operation that allows several users to share a database.

node A particular CPU with its associated storage.

null value The absence of a value for a given column. A null value should mean only that nothing is known about the value. Columns can required to contain values by defining them as NOT NULL at table creation.

ODL An ORACLE utility used to load data from operating system files into ORACLE database tables. Will be replaced by SQL*Loader.

online backup The archiving of database data while the database is running. Thus, the DBA need not shutdown the database to archive data; in fact, even data which is being accessed can be archived.

offline redo log When using the redo log in ARCHIVELOG mode, a log file which has already been archived.

online redo log Redo log files which have not been archived, but are either available to the instance for recording activity or have already been written and are waiting to be archived.

open database A database connected to an instance and available for access by users.

optimizer The part of the ORACLE kernel which determines the best way to use the tables and indexes to perform the operation requested by a SQL statement.

ORACLE Program Interface (OPI) The ORACLE half of the Program Interface, which provides security between the user program and the ORACLE program. See also *User Program Interface (UPI)*.

parameters (INIT.ORA) A set of standard names and values that are listed in the INIT.ORA file and are read when an instance is started in order to build the SGA. These parameters can be used to tune the performance of a database system.

parse The examination and validation of a SQL statement to make sure it is properly formed and that all necessary information is supplied, so that the RDBMS can execute it. A parse error usually occurs because of incorrect syntax, or naming of non-existent database objects.

parse lock One of two types of dictionary locks, acquired on behalf of users who are referencing tables in SQL statements. Parse locks are shared. Contrast to DDL locks, the other type of dictionary lock.

PCTFREE The ratio of space in each data block to be reserved for updates to rows stored in that data block.

PCTINCREASE The percent by which a subsequent extent of blocks is larger than the previous extent of blocks.

PECTUSED The percent of space in a data block which ORACLE will attempt to keep filled.

PL/SQL A procedural extension of SQL that provides programming constructs like blocks, conditionals, and procedures.

PMON process One of five background processes used by multi-process database systems, the Process Monitor process performs recovery when a process accessing a database fails.

port Term used to refer to any individual operating system which supports the ORACLE RDBMS. While much information regarding ORACLE is true across all ports, some information necessarily varies depending on the operating system; this information is usually called *port-specific*, or *operating system dependent*.

precision The number of digits which are actually saved and considered significant by the database. Users may set precision for datatype NUMBER when creating tables with numeric columns.

predicate A portion of a SQL statement, the WHERE clause, that imposes a set of criteria on the data to be returned by a query. See also *expression*.

procedure A stored set of instructions (typically SQL statements and PL/SQL commands) saved in PL/SQL for repeated execution.

process Essentially a synonym for "user", a single entity having access to a database and performing a set of tasks. In addition to user processes, however, there are also up to five background processes that keep the database running.

private rollback segment A rollback segment whose name has been specified in the INIT.ORA file for a particular instance, and is thus reserved for that instance.

primary key See *key*.

program global area (PGA) Memory dedicated to a particular user of the database. Contrast with *system global area*.

program interface A procedural interface to the database that provides access to the RDBMS for user application programs.

public rollback segment A rollback segment that may be allocated by any instance accessing a database. Contrast with *private rollback segment*.

query A frequently used type of SQL statement, used to retrieve data. Queries typically begin with the SQL reserved word SELECT.

quota A resource limit. Quotas can limit the amount of storage consumed by each user of the database.

RAW data One of the standard ORACLE datatypes, raw data may contain data in any form, including binary.

RDBMS Acronym for Relational Database Management System.

read consistency A feature of ORACLE RDBMS whereby the results returned by every query are guaranteed to be a consistent set of data as of the time the query was executed.

recursive calls A nested invocation of the RDBMS that occurs when one SQL statement requires the execution of other SQL statements. For example, auditing information is recorded in system tables using a recursive call.

redo log A sequential log of actions which are to be reapplied to the database if the changes were not written to disk. The log consists of at least two files; one is optionally being spooled while the other is being written. When the one currently being written fills, the next one is reused.

redo log sequence number A number used to identify a redo log, used when applying the log files for recovery.

remote database A database on a computer other than the local one. Usually a computer on the same network, but at a different node.

repeatable read A feature in which multiple subsequent queries return a consistent set of results (as though changes to the data were suspended until all the queries finished).

reserved word One of a number of words which has special meaning to the ORACLE RDBMS, and therefore cannot be used to name any database object. Examples are TABLE, NUMBER, DATE, SELECT.

resource A general term for a logical database object or physical structure which may be locked. Resources that users can directly lock are rows and tables; resources that the RDBMS can lock are numerous and include data dictionary tables, caches, and files.

revision level The third identifying number of ORACLE software. In V6.0.10, the revision level is 10.

roll back When a user enters the SQL command ROLLBACK, he undoes all the changes done in the current transaction.

rollback segment A set of entries used to undo changes in the database in the event of transaction rollback, crash, media recovery, or if necessary for read consistency. Every instance requires at least one dedicated rollback segment.

roll forward The reapplying of changes to the database. Necessary for media recovery and sometimes for instance recovery. The redo log contains the redo entries used for roll forward.

row cache An area in the SGA used to store rows of frequently used dictionary information in order to speed parsing.

row header The portion of each row that contains information about the row other than row data, such as the number of row pieces, columns, etc.

ROWID A sequence of characters that uniquely identifies both a row and the table that contains the row.

row level lock manager The portion of the kernel which allows locking on the row level rather than the table level, thus allowing a high level of concurrency and transaction throughput.

ROWNUM An index starting with 1 that identifies the position of a row within a table.

row piece A portion of a row, containing row header information and row data.

savepoint A point at which all work in a transaction is saved. A transaction can use a series of savepoints as intermediate points for either committing or rolling back to.

scale A user defined limit on the number of decimal places to be stored for a column of datatype NUMBER. Indicated when a table is created.

segment A collection of blocks reserved for a table's data (rows), data for a single index, or for a rollback segment. A table will always have one data segment and as many index segments as indexes. A cluster will always have at least two segments (one data and one cluster index).

segment header block The first block in the first extent of a segment, which contains, among other things, a list of extents for this segment.

SELECT list The list of columns or expressions, that follow the keyword SELECT in a SQL query.

sequence A database object created like a table that is used to generate unique keys (sequence numbers). Sequence information is stored in the data dictionary.

server The provider of services requested by a client.

server processes Processes that work on behalf of user processes. Multiple user database systems use the four background server processes DBWR, LGWR, SMON, and PMON, and sometimes ARCH.

session The duration of time from one username connecting to and disconnecting from the database.

share lock A type of lock which allows the same type of lock to coexist on the locked resource, so that more than one user or process has access to that resource.

shared disk system Multiple instances on one or multiple CPUs that share a common database.

shut down The process of disconnecting an instance from a database and terminating the instance. Contrast with *start up*.

single-process A mode of database operation that allows only one user to access a database.

SMON process One of five background processes used by multi-process database systems, the System Monitor process performs recovery when a instance accessing a database fails.

SQL The standard name for the Structured Query Language, pronounced "sequel." See

also *Data Manipulation language*, *Data Definition language*, and *Data Control language*.

SQL*DBA An ORACLE utility included with the ORACLE RDBMS V6, for use by DBAs while performing database maintenance and monitoring.

SQL*Loader An ORACLE tool used to load data from operating system files into ORACLE database tables.

start up The process of starting an instance, presumably with the intention of mounting and opening a database, in order to make a database system available for use.

statement (SQL) A "complete sentence" in SQL. Most statements are *executable SQL statements* but not all are. Portions of a statement are called *expressions*, *clauses*, or *predicates*.

SYS user One of two standard DBA users automatically created with each database. This user owns the base tables for the data dictionary.

SYSTEM user One of two standard DBA users automatically created with each database.

system global area (SGA) Memory shared by all users of a database. Provides communication between the user and the background processes.

system privileged SQL*DBA commands A subset of the SQL*DBA commands which require not only access to the SQL*DBA utility, but a special operating system account. These commands require the highest level of security.

tablespace A logical portion of a database used in allocating storage for table data and table indexes.

tablespace locks One of three types of internal locks, these locks exist on tablespaces and rollback segments to make sure that instances properly access and write to tablespaces and rollback segments.

temporary segment A segment required by the RDBMS for processing, temporarily created and allocated to a user process, and which disappears when the process no longer needs it. A temporary work area, typically used for sorting.

transaction lock A lock appearing in the MONITOR display indicating that a transaction is holding row locks.

transaction processing option An option available with ORACLE RDBMS Version 6 intended for systems designed to handle many transactions with a high degree of concurrency.

TT lock A dictionary lock on a temporary segment.

used extents Extents which have been allocated to either a data (table) segment or an index segment, and thus have data in them, or have been reserved for data.

username The name by which a user is known to the ORACLE database and to other users, as in the prefix to a table name (as in SCOTT, in table SCOTT.EMP). Every username is associated with a secret password, and both must be entered and verified by the data dictionary, in order to connect to ORACLE.

data dictionary, in order to connect to ORACLE.

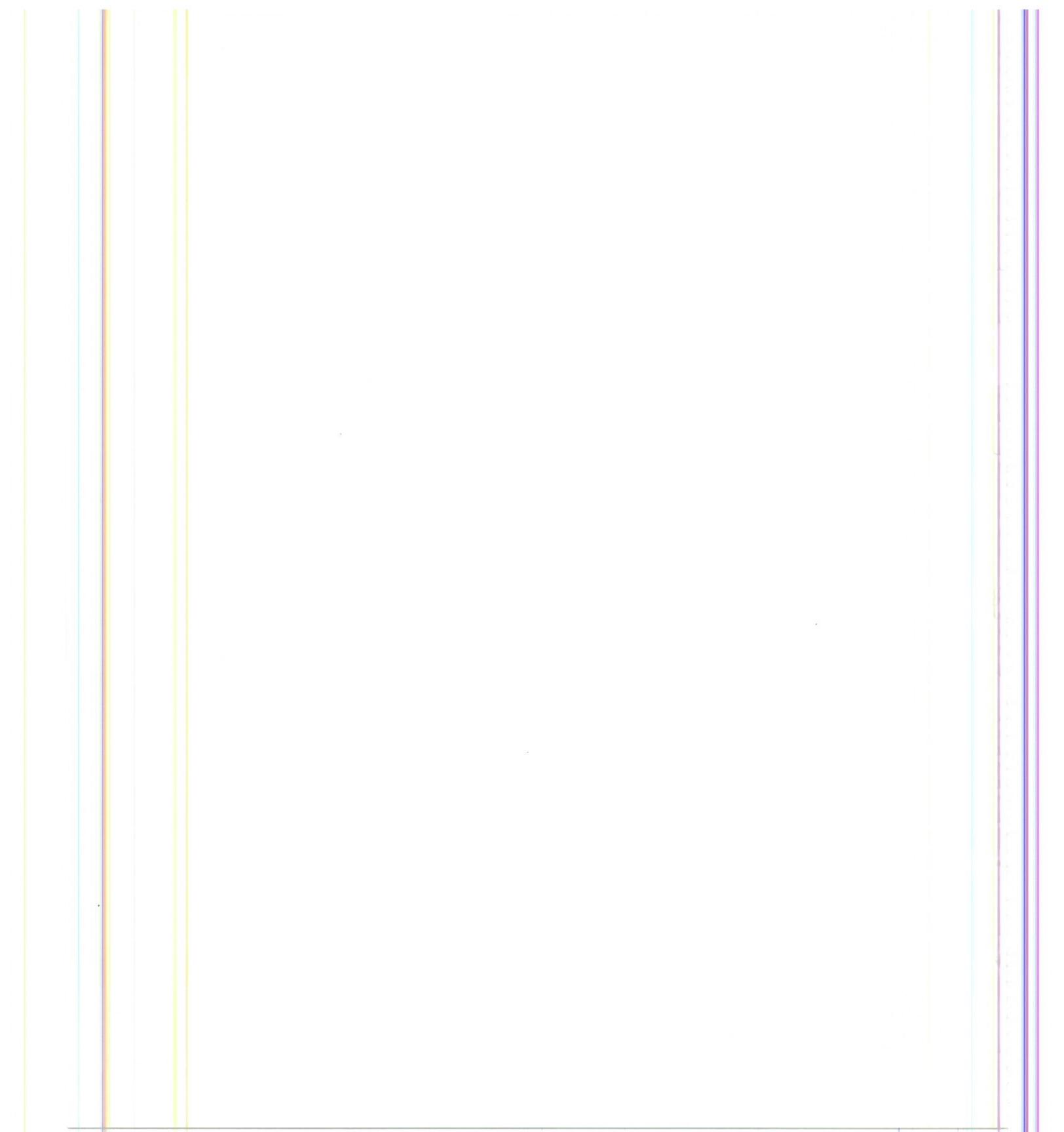
User Program Interface (UPI) The user half of the Program Interface, which is responsible for keeping a separation between the user program and the ORACLE kernel for data security.

warm start See *start up*.

variable parameters The INIT.ORA parameters whose values may range widely and whose values may greatly affect the performance of a database. These are the values that DBAs may wish to "tune" or experiment with, to optimize their database's performance.

version number The primary identifying number of ORACLE software. In V6.0.10, 6 is the version number.

view A database object that can often be treated just like a table. A view is defined by storing a SQL query, and whenever the view is queried, the stored query is executed against the *base tables* upon which the view was defined.



INDEX

Special Characters

24 hour format 6-4

A

Abbreviations for storage 4-9
Abnormal process termination 4-15,11-3
Abnormal transaction termination 5-25
ABORT mode B-41
ACCESSIBLE_COLUMNS view 7-5, E-2
ACCESSIBLE_TABLES view 7-5, E-2
Access path 1-7, 5-11, 10-3
 choosing optimal 10-6
 using cluster key 19-16
 using concatenated index 19-16
 using ROWID 19-16
Access, database 17-2
 monitoring B-10
Access, database object 1-6,10-3
 checking for proper 10-3
Access, SQL*DBA 14-4
Accounts, operating system 2-3
 privileges required by DBA 13-3
Acquiring locks 12-5
Adding files 3-5
Addresses, row 6-8
Addressing exceptions 15-4
AIJ utility (V5) A-2
ALL data dictionary views 7-4
ALL_CATALOG view 7-5, E-2
ALL_COL_COMMENTS view E-2

ALL_COL_GRANTS view 7-5
ALL_COL_GRANTS_MADE view E-2
ALL_COL_GRANTS_REC'D view E-3
ALL_DB_LINKS view 22-9, E-3
ALL_DEF_AUDIT_OPTS view E-3
ALL_INDEXES view E-4
ALL_IND_COLUMNS view E-4
ALL_OBJECTS view A-3, E-5
ALL_SEQUENCES view 5-23, E-5
ALL_SYNONYMS E-5
ALL_TABLES view E-5
ALL_TAB_COLUMNS view 7-5, A-3
ALL_TAB_COMMENTS view E-6
ALL_TAB_GRANTS_MADE view E-6
ALL_TAB_GRANTS_REC'D view E-6
ALL_TAB_GRANTS view 7-5
ALL_USERS view E-7
ALL_VIEWS view E-7
Allocations, redo log file 20-18
ALTER CLUSTER statement 4-11, 5-4, 16-11
ALTER DATABASE 3-9, 14-3,14-5, 14-10,
15-15, 16-3
 changing redo log mode 15-9
 OPEN option 13-6
 redo log maintenance 16-3
 RENAME option 16-4
 syntax G-2
ALTER INDEX 4-11, 16-14
ALTER ROLLBACK SEGMENT 16-6
 syntax G-5
ALTER statements 1-5
ALTER TABLE 4-11, 5-4, 16-13

ALTER TABLESPACE statement 3-6, 4-7, 4-11, 14-7, 16-7 - 16-10
 backup 15-23, 15-25, 15-26
 changing default storage 16-13
 syntax G-6

ALTER USER statement 4-18, 5-2, 16-11, 17-5
 changing tablespace for temporary segments 16-17
 syntax G-8

American National Standards Institute (ANSI) 1-3
 data dictionary views 7-5

ANSI compatible mode 12-20

Applications
 grouping tables 4-5
 temporarily unavailable 4-6
 tuning individual 20-4
 upgrading while offline 4-6

ARCH background process 1-11, 9-8
 starting B-27
 trace files generated by 15-20
 use in archiving 15-19

Archive destination 15-20

ARCHIVE LOG command 15-19 - 15-20, B-26

ARCHIVE_LOG_DEST B-27

ARCHIVELOG mode 3-8, 3-10, 9-8, 14-3, 15-8 - 15-11, 15-13, 15-18, 15-24, 16-2 - 16-3
 checkpoints and 15-12
 description of 15-10
 dropping online log files 16-4
 requirements 3-10
 switching to 15-15
 vs. NOARCHIVELOG mode 3-9

Archiver (ARCH) process
 see ARCH background process

Archiving redo log files 2-6, 3-8, 15-8, 15-12, 15-18, B-26
 archive location B-27
 automatic vs. manual 9-8, 15-10
 checkpoints before 3-12
 device written to 15-19
 displaying files for archive 15-19
 enabling 9-8
 errors during 15-20
 frequency of 3-11, 16-3
 locks to protect 12-17

prerequisites for 14-3
 re-archiving B-27
 setting mode of 15-9
 stopping B-27
 trace file records of 15-19

ARH process (V5) A-10

Array processing 10-7, 19-1, 19-17

Arrays
 size 10-8
 storing 6-5

ASC index option 5-15

ASCII data 6-2

Assignments 1-6

Asynchronous drivers 9-9

AUDIT_ACTIONS view 7-5, E-7

AUDIT statement 1-6, 12-16, 17-10, 18-4
 syntax G-9, G-11

AUDIT_TRAIL table 17-10, 17-12, 18-4
 deleting rows from 7-10, 18-8

AUDIT_TRAIL parameter 17-10, D-6

Auditing 20-3
 affect on performance 20-23
 enabling 17-10
 options in effect 17-11
 privileges to perform 17-4
 purpose of 17-10
 system operations 17-11
 system-wide defaults 17-10
 use of sequence numbers 5-26

Auditing, table
 default options 18-6
 listing current 18-7

Automatic archiving 15-10, 15-19
 disabling 15-20

Automatic login usernames 17-7

Automatic startup 1-10

AVAILABLE rollback segments 20-21

B

B*-tree indexes 5-17
 advantages of 5-18

Background process failure 15-7
 repeated 15-7

Background processes 1-11, 9-6, B-34
 absence in single-process systems 9-6

failure of 15-4
 PGA size for 8-4
 starting 14-5
 startup after abnormal termination B-41
BACKGROUND_DUMP_DEST parameter 15-7, D-6
 Backups 1-7, 2-3, 3-10, 4-5, 14-2, 15-8
 closed or open databases 15-22
 consistent set of files 15-27
 database 3-10, 14-4, 15-8, 15-10, B-1
 difference from Export 15-30
 entire database 15-22
 grouping tables for 4-5
 individual tablespaces 15-22
 restoring tablespace 15-25
 tablespace 15-22
 software required 15-2
 uses for 15-21
 using closed database 15-22
 Batching factor 20-16, 20-24
 Before image file (V5)
 replaced by rollback segments A-5
 Bert and Ernie 12-4
 Binary data
 storing with LONG or LONG RAW 6-6
 Bind variables 10-4, 10-7
 not valid for DDL 10-8
 using new values with same datatype 10-4
 BIW process (V5) A-10
 Blanks
 storing trailing 6-2
 Block size
 changing 3-5
 database file 3-5
 default 3-5
 effect on PCTINCREASE 4-10
 ORACLE vs operating system 3-5, 4-14
 Blocks, ORACLE
 accessing desired one in buffer cache 20-11
 allocated but empty 6-9
 available 5-4
 currently holding data 6-9
 definition of 3-5
 format 5-1, 5-3
 forming extents 4-8
 in ROWID 6-8
 reclaiming 4-11
 percent utilized 16-11
 Boolean expressions 6-10
 Bootstrap segment 4-1, 4-3, 4-8, 4-19
 Bottlenecks 20-23
 Bounded range query path 19-11, 19-16
 Branch index blocks 5-15, 5-17
 Buffer busy waits statistic 20-12, 20-13
 Buffer cache 20-11
 modified blocks in 20-11
 monitoring 20-12
 rollback segments in 20-22
 Buffer manager 20-4, 20-11
 Buffers, database 3-5, 8-3
 cleaned by DBWR 20-12
 cleaning to make free 20-14
 DBWR free needed statistic 20-14
 dirty 20-14
 free buffer waits statistic 20-15
 LRU list 20-11
 number of 20-12
 optimizing I/O to and from 20-6
 running out of free 20-14
 searched by user process 20-15
 Buffers, log
 writing from 9-7
 Bug fixes 1-8
 BWR process (V5) A-10

C

Cache buffer chains latch statistic 20-14
 Cache buffer LRU chain latch statistic 20-14
 Cached data 20-12
 CALLS parameter D-6
 Cartesian products 19-16
 CATALOG view (V5) A-3
 CATALOG5.SQL file A-3
 CCF utility (V5) A-2, A-9, A-10
 Central control (distributed databases) 22-8
 Chained cluster blocks 5-22
 Chained rows 5-9
 Changing passwords
 DBA users 2-4
 CHAR datatype 6-1 - 6-2, 6-7 - 6-8
 compared to RAW 6-6

description of 6-2
displaying AB-38
maximum characters 6-2

Character set
allowable 6-2
argument to LANGUAGE F-2
CONVERT function F-8
foreign language F-2
problems in translating F-6
supported acronyms D-17

CHARWIDTH
setting B-37

CHECK clause A-8

Checkpoint interval 15-16

CHECKPOINT_INTERVAL parameter 3-11

Checkpoints 3-12, 15-11, 20-16
affect on performance 20-17
frequency of 15-12 - 15-13, 15-15, 15-16
information in redo log 3-9
initiating 15-11
multiple 15-11
NOARCHIVELOG mode and 15-12
number of blocks written to trigger 15-16
purposes of 15-12
reducing frequency of 20-6, 20-16 - 20-17
updating control file 3-7

Chunk allocations statistic 20-17 - 20-18

CLEANUP_ROLLBACK_ENTRIES parameter D-6

Client nodes 22-5
dblinks 22-9
individual accounts 22-11
one central account 22-11
public DBlinks in 22-10

CLN process (V5) A-10

Closed database
for redo log file maintenance 16-3

Closing a database 1-10, 14-10

Closing cursors 8-7

Cluster blocks
chained by key 5-22

Cluster columns
restrictions 5-20

Cluster indexes 5-20, 5-21
allocating to tablespaces 4-12
average size of all entries for 16-15

concatenated 5-21
creation 5-13, A-6
estimated bytes per 5-22
ideal 5-21
indexing 4-12
nulls in 6-7
restrictions 5-20
segments for 4-12
vs. cluster index 5-20

Cluster keys 5-3, 5-19, A-6
chained values for one 5-5
choosing 5-20, 16-15
information in row headers 5-9
per data block 5-3, 5-5
too large 16-15
too much or too little data per 5-21
too small 16-15
updating 5-20

Clusters 5-1, 5-19, 16-15
adding tables 5-21 , 16-16
block format 5-3
choosing tables for 5-21
creating 5-21, 6-7
dictionary locks on 12-16
directory of tables 5-3
dropping 16-17
effect on storage 5-19
importing 5-22
lack of index 5-20
loading 5-13, 16-16
monitoring space used by 16-16
rows per block 5-22
single tables 5-21, 16-15
SIZE option 5-22
space required by 16-1, 16-15
unavailable 5-21
using effectively 19-1
& compared to V5 A-6
when recommended 5-19, 16-15, 19-11

CLUSTERS view (V5) A-3

Code, ORACLE
executable 8-2
owner of 3-3
shared by instances 8-2
storing program 8-2

COL view (V5) A-3
Column data
 guaranteeing unique 5-12
 length 5-9, 5-10
 spanning blocks 5-9
 trailing blanks 5-9
 when null data is stored 5-9
Column names 1-4
COLUMN_PRIVILEGES view 7-5, E-7
Columns 1-4
 clustering 5-21
 datatypes 5-2
 definitions validated during parsing 10-6
 limiting access to certain 18-2
 maximum number per table 6-2
 names 5-2
 optional or mandatory 5-2
 order in index 5-13, 5-15
 order of storage 5-9
 overhead 5-10
 restricted UPDATE access 18-3
 revoking privileges from select 18-4
 versus ROWID 6-9
 width 5-2
COLUMNS view (V5) A-3
COMMENTS columns 7-3
COMMENT statements 12-16
COMMIT WORK statement 1-5, A-7
 explicit and implicit 11-9
 omission before process termination 11-3
 results of 11-5
Committed work 15-5
Committing work 9-7, 15-11, 16-2, 20-2,
 batching for writes 20-16
 file I/O 20-11
 LGWR notified 20-16
 skipped sequence numbers 5-25
 vs. successful execution 11-2
Common header (data block) 5-3
Communications
 between processes 9-8
 between user and ORACLE in program interface 9-9
 part of program interface 9-9
 protocols 22-2
Composite primary keys 5-13
COMPRESS index option 5-15
Compression, index A-6
Concatenated indexes 5-13, 19-7
 leading columns 5-13
 queries on leading columns 19-7
Concurrency 12-1, 12-21
 managing 1-7
 primary concerns 12-2
Conditional control 1-6
CONNECT BY clause 5-25
 LONG data in 6-6
CONNECT command B-28
CONNECT INTERNAL 14-4
CONNECT privilege 17-3
Connected users
 at shutdown 14-10
Connecting to a database 9-3, 9-9, 17-3, B-2,
 B-28
 allocating PGA 8-4
 auditing 17-11
 monitoring unsuccessful 17-10
Connections
 inhibited during shutdown 14-10
 limited to DBAs 14-6
 multiple and two-task 9-4
Consistency, data 12-5, 12-8
 across multiple tables 12-11
Consistent backup 15-22
CONSTRAINT_COLUMNS view 7-5, E-8
CONSTRAINT_DEFS view 7-5, E-8
Contention
 block-level 20-13
 buffer busy waits statistic 20-13
Context areas 8-1, 8-5
 increment allocations 8-6
 initial space allocation 8-5
 maximum extents 8-6
 maximum number per process 8-5
 resetting size 8-7
 size of 8-5
 versus cursors 10-2
CONTEXT_AREA parameter 8-5, 8-7, D-7
CONTEXT_INCR parameter 8-6 - 8-7, D-7
Contiguous allocations, file 3-4, 20-5
Control files 1-10, 3-6, 16-1, A-4
 at startup 14-5

automatic or manual creation 13-2
backing up 15-22
checkpoints and 15-12
contents 3-7
copying 15-28
creation 13-2, 13-6
device allocation 15-13, 20-10
failure reading or writing 15-8
impact of MAXDATAFILES on size of 3-4
impact of MAXLOGFILES on size of 3-11
locks on 12-17
media failure to 15-28
names 3-7
names required at startup 14-8
number of 3-7, 15-13
optimal placement 20-10
recovering 15-4
specifying at database creation 13-3
writing of 3-7
CONTROL_FILES parameter 3-7, 13-3, D-7
Conversion, lock 12-14
CONVERT function F-8
Converting data 6-5, 6-10
COPY command (SQL*Plus) 22-8
 LONG data in 6-6
Correlated subqueries 4-18
CPU capacity 20-2
CPU failure 15-7
CPU-bound systems 20-23
CPU_COUNT parameter D-7
CREATE CLUSTER statement 4-11, 5-4, 5-20, 16-11
 SIZE option 16-15
 storage parameters 5-2
CREATE DATABASE 1-10, 2-6, 3-4, 3-5 - 3-7, 3-9, 4-3, 13-1 - 13-5, 14-2, 15-13, 15-14, 16-5, A-2
 choosing redo log mode 15-9
 database name 4-2
 initial rollback segment 15-16
 LOG_FILES option 15-14
 MAXLOGFILES argument 3-11
 on existing database 13-22
 specifying database name 1-11
 syntax G-13 - G-15
CREATE INDEX 4-11, 4-18, 5-14 - 5-15, 19-13
 cluster index 5-20
 indicating tablespace 20-10
CREATE PUBLIC SYNONYM statement 17-14
CREATE ROLLBACK SEGMENT 16-5
 syntax G-16
CREATE SEQUENCE statement A-5, A-8
CREATE statements 1-5
 transaction entries 5-4
CREATE SYNONYM statements 12-16
CREATE TABLE statement 5-2, 5-4, 5-10, 16-11, 16-16
 CLUSTER option 5-20
 default storage parameters 5-2
 indicating tablespace 20-10
 locks required 12-16
CREATE TABLESPACE statement 4-7 - 4-8, 4-11, 16-2, 16-7, 16-8, 16-9, 16-11
 syntax G-17 - G-18
CREATE VIEW statements 12-16
Creating a database
 see CREATE DATABASE statement
CRT program 1-7, B-2
Cumulative exports 15-28, 15-29
 defined 15-31
Current activity
 user processes 8-4
Current state of query 8-2
CURRVAL 5-25
 used with NEXTVAL 5-25
Cursor ID 10-2
Cursors 8-5, 19-17
 automatic creation and use 10-5
 closing 8-7
 closing at disconnect B-30
 closing or deallocating 10-6
 creating 10-3
 explicit creation of 10-5
 maximum number per user 10-5
 maximum open 8-5
 opened at CONNECT B-28
 resetting size 8-7
 reusing A-7
 status information 8-5
 versus context area 10-2
 when to increase number of 10-5
CYCLE (MONITOR)

setting B-5, B-37
valid range for B-38

D

Data 3-4, 4-2

COPY command and remote 22-8
distributed across nodes 22-6
in cluster to be dropped 16-17
in tablespace to be dropped 16-10
location of 22-3

separating table from index 20-7
storing locally or remotely 22-8
temporary 8-2

Data "snapshots" 12-3

Data blocks A-6

Data consistency 2-3

Data Control statements 1-6

Data conversion 6-5, 6-10, 10-7
errors in 10-6

Data Definition statements 1-5

Data dictionary 1-11, 2-5, 2-6, 3-5, 4-3, 6-5,
7-1 - 7-10

adding new objects to 7-9

accessing 7-2, 7-3

altering 7-10

auditing information 18-4

base tables 2-4, 7-3

caching 7-9

cluster information 16-16

creation 7-2, 7-8, 13-4, 13-6

DBlinks stored in 22-9

DDL statements requiring write access to 10-8

definition of 7-2

deleting from 7-10

equivalence table A-3

extending 2-6

extent information 4-12

in distributed databases 22-4

locking 12-6, 12-12, 12-16

online tablespace information 4-6

public synonyms for 7-8, 17-14

quota information 17-6

referential integrity information A-8

rollback segments 16-6

summary of views 7-8

synonyms 7-7

tables for other Oracle products 13-7

tablespace information 16-8

temporary segment information 16-17

updating 7-2

user information 17-2 - 17-3

validations during parsing 10-6

Version 5 A-3,

views 2-4

Data integrity 2-3

Data locks 12-7

on rows (TA) 12-18

overriding 12-11

purpose of 12-8

TT table lock 12-18

Data Manipulation statements (DML) 1-5

Data redundancy 22-8

Data schema 19-2, 20-4

Data security 2-2

Data segments 4-1, 4-8, 4-12

minimizing extents in 20-8

reclaiming space 4-11

Data sharing 2-3

Data values

substituting in SQL statements 10-7

Database

backup files for 15-8

definition of 1-10, A-2

enlarging 16-7

mounted but not open 15-15, 15-26

mounting B-1

opening B-1

reading 20-14

shared by instances 1-12

taking portions offline 4-6

Database access 1-6

Database Administrator (DBA)

access to account 2-3

creating additional 17-5

database privileges 2-6

limiting database access to 14-6

multiple 2-4, 17-4, 22-6

operating system account 2-3

responsibilities 2-2

SYS vs. SYSTEM 2-4

system privileged SQL*DBA commands 17-5

Database backup
 see Backups

Database blocks
 reading into buffer cache 20-11

Database buffers 15-5
 number of 3-5
 size of 3-5
 writing during checkpoint 15-12
 writing from 9-7, 11-6
 writing to database 3-12, 15-11

Database creation
 see CREATE DATABASE

Database design 19-2

Database exports 15-30

Database files 1-10, 3-4, 4-2, 13-2, 16-1, A-4
 adding 3-5, 4-7, 16-7 - 16-8
 allocating 3-4
 and tablespaces 16-2
 at startup 14-5
 automatic or manual creation 13-2
 backing up 15-22
 blocksize 3-5
 contiguous requirements 20-5
 corresponding to SYSTEM tablespace 3-4
 creating for ORACLE use 13-2
 definition of 3-3
 distributing I/O across 4-4
 dropping 16-2
 enlarging 16-2
 failure in non-SYSTEM tablespace file 15-8
 failure in SYSTEM tablespace file 15-8
 failure reading or writing 15-24
 information in redo log 3-9
 locks on 12-17
 minimum size 3-5
 monitoring activity with MONITOR B-25
 naming 3-7, 16-8
 number of 3-4, 3-11
 offline and online 3-6
 on different device than redo log files 15-15
 20-6, 20-10
 original 13-6
 physical reads/writes 20-7
 preparing 3-4
 recovering 15-4, 15-24

removing 3-6
renaming 16-9
requirements of 3-4
restoring backups of 15-25 -15-26
ROWID and 6-8
spanned by segments 4-8
temporarily unavailable 3-6
writing modifications to 15-5

Database links
 see DBlinks

Database name 3-7, 4-2, 14-5, B-43
DBlink 22-9

Database name (SQL*Net) 22-7

Database names 1-11

Database privileges
 granting to client account 22-11

Database recovery 15-28

Database specifications B-4

Database system 1-9, A-2

Database Writer (DBWR) process
 see DBWR background process

Databases
 offline 3-1
 accessing remote 14-4
 availability 22-4
 choosing the default 22-7
 connecting with DBlinks 22-9
 dismounted 14-3
 distributed 22-3
 mounted but not open 14-3
 mounting shared or exclusive 14-3
 multiple 1-13
 open 14-3
 relationship among distributed 22-4
 steps for accessing 14-2
 treating non-ORACLE as ORACLE 22-2

Datatypes 1-4, 1-11
 conversions 6-10
 mixing 6-10
 SQL/DS or DB2 6-7

DATE data
 default 6-4
 displaying B-38
 non-English names F-2

DATE datatype
 julian 6-5

using Julian in calculations 6-5
DATEWIDTH
 setting B-37
Day names
 using local language F-2, F-4
Day numbering F-5
DB2 databases 22-2
DB2 datatypes 6-7 - 6-8
DB_BLOCK_BUFFERS parameter 3-5, 20-12,
 D-7
 other parameters relying on 20-13
 why increase 20-6
DB_BLOCK_CACHE_PROTECT parameter
 D-7
DB_BLOCK_HASH_BUCKETS parameter D-7
DB_BLOCK_MAX_CLEAN_PCT parameter
 20-13 - 20-15, D-8
DB_BLOCK_MAX_MOD_PCT parameter
 20-13, 20-14, D-8
 finding optimal value 20-15
 reducing 20-15
DB_BLOCK_MAX_SCAN_PCT parameter D-9
DB_BLOCK_MULTIPLE_HASHCHAIN_
 LATCHES parameter D-9
DB_BLOCK_SIZE parameter 3-5, 4-9, 13-3,
 D-9
DB_BLOCK_TIMEOUT_WRITE_PCT
 parameter 20-13 - 20-14, D-9
DB_BLOCK_WRITE_BATCH parameter D-10
DB_FILE_MULTIBLOCK_READ_COUNT
 parameter D-10
DB_FILE_SIMULTANEOUS_WRITES
 parameter D-10
DB_FILES parameter 3-4, 3-11, 8-4, D-11
 effect on control file 3-7
DB_HANDLES parameter D-11
DB_HANDLES_CACHED parameter D-11
DB_NAME parameter D-11
DBA data dictionary views 7-4
DBA mode 14-4, 14-6, B-43
DBA privilege 2-3, 17-4, B-43
DBA responsibilities
 distributed 22-4
DBA statements
 auditing of 17-11
DBA-only access to database 14-4, 14-6
DBA_AUDIT_DBA view E-8
DBA_AUDIT_EXISTS view E-9
DBA_CATALOG view E-9
DBA_CLUSTERS view E-10
DBA_CLU_SOLUMNS E-10
DBA_COL_COMMENTS view E-10
DBA_COL_GRANTS view E-10
DBA_CROSS_REFS view E-11
DBA_DATA_FILES view E-11
DBA_DATA_SEGMENTS view 16-8
DBA_DB_LINKS view 22-9, E-11
DBA_EXP_FILES view E-11
DBA_EXP_VERSION view E-11
DBA_EXTENTS view E-12
DBA_FREE_SPACE view E-12
DBA_INDEXES view E-12
DBA_IND_COLUMNS view E-13
DBA_RB_SEG_STATUS view 16-6
DBA_ROLLBACK_SEGS view 16-6, E-13
DBA_SEGMENTS view E-13
DBA_SEQUENCES view E-14
DBA_SYNONYMS view E-14
DBA_SYNONYMS.SQL file 7-5
DBA_SYS_AUDIT_OPTS view E-14
DBA_TABLES view E-15
DBA_TABLESPACES view E-15
DBA_TAB_AUDIT_OPTS view E-15
DBA_TAB_COLUMNS view E-16
DBA_TAB_COMMENTS view E-16
DBA_TAB_GRANTS view E-17
DBA_TS_QUOTAS view E-17
DBA_USERS view E-17
DBA_VIEWS view E-18
DBAs
 see Database Administrator (DBA)
DBlinks 22-5
 client end 22-9
 database name 22-9
 example 22-9
 individual user 22-12
 network protocol 22-9
 nodename 22-9
 private 22-10
 protocol options 22-9
 public 22-10
 server end 22-9

username/password 22-9
using synonyms to mask 22-9
DBWR background process 1-11, 4-6, 9-7, 20-4, 20-11, 20-13
actions during checkpoint 20-17
contention with user processes 20-10, 20-15
notified to begin cleaning 20-13
notifies LGWR 20-16
optimal work level 20-14
periodic wakeups 20-14
DBWR buffers scanned statistic 20-14 - 20-15
DBWR checkpoints statistic 20-17 - 20-18
DBWR free low statistic 20-14
DBWR free needed statistic 20-14 - 20-15
DBWR timeouts statistic 20-14
DBWR-bound system 20-23, 20-24
DCL statements 1-6, 11-3
DC_COLUMN_GRANTS parameter D-11
DC_COLUMNS parameter D-11
DC_CONSTRAINT_DEFS parameter D-11
DC_CONSTRAINTS parameter D-12
DC_FILES parameter D-12
DC_FREE_EXTENTS parameter D-12
DC_INDEXES parameter D-12
DC_OBJECT_IDS parameter D-12
DC_OBJECTS parameter D-12
DC_ROLLBACK_SEGMENTS parameter D-12
DC_SEGMENTS parameter D-12
DC_SEQUENCE_GRANTS parameter D-12
DC_SEQUENCES parameter D-12
DC_SYNONYMS parameter D-13
DC_TABLE_GRANTS parameter D-13
DC_TABLES parameter D-13
DC_TABLESPACE_QUOTAS parameter D-13
DC_TABLESPACES parameter D-13
DC_USED_EXTENTS parameter D-13
DC_USERNAMES parameter D-13
DC_USERS parameter D-13
DDL_LOCKS parameter D-13
DDL locks 12-7, 12-16
DDL statements 1-5, 11-3, 11-8
automatic commits 11-3, 11-9
execution 10-8
locks released 12-17
not requiring DDL locks 12-16
remote 22-6
requiring DDL locks 12-16
Deadlocks 10-6, 11-8, 12-14, 12-19
 avoiding 12-20
DECIMAL datatype 6-7
Declaring variables 1-6
DECNET 9-9
DECODE SQL function 19-4
DEFAULT clause A-8
Default databases 22-7
DEFAULT STORAGE 16-9
Deferred writing 20-2
Define phase of execution 10-7
 not valid for DDL 10-8
Defining database objects 1-5
Definitions, database objects
 dictionary locks to protect 12-16
 preventing change during references to 10-4
DELETE statements 1-5
 read consistency 12-8, 12-10
 remote 22-6
Describe phase of execution
 purpose of 10-7
Destination
 archive B-27
Detached processes
 see Background processes
Devices
 failure in 15-8
 using tablespaces to allocate 4-3
Devices, storage 4-6
Dictionary cache locks 12-17
Dictionary caches 8-3
 locks on 12-17
Dictionary locks 12-7, 12-11, 12-13, 12-16
 exclusive (DDL locks) 12-16
 on temporary segments (TT locks) 12-18
 shared (parse locks) 12-16
 shared or exclusive TD locks 12-18
Dictionary tables 1-10
DICTIONARY view 7-3, A-3, E-18
DICT_COLUMNS view 7-5, E-18
Direct memory access (DMA) 20-5
Directory structures 20-5
Disconnect 11-3, B-30
 abnormal 15-4
 commit at normal 11-9

DISCONNECT command B-28, B-30
Disk backup 15-10, 15-21
Disk drives
 allocating redo and database files 15-15
 contention 4-5, 15-15, 20-10
 errors 4-7
Dismounted databases 14-3
Dismounting a database 14-10
DISTINCT clause 19-13, 19-5
Distributed databases 1-12 - 1-13, 22-2
 COPY command 22-8
 DBA concerns related to 22-6
 definition of 22-3
 designing 22-7
 differences from distributed processing 22-4
 linking database nodes 22-5
 relationships of databases 22-4
Distributed processing 22-2
 definition of 22-2
 differences from distributed databases 22-4
 restrictions on 22-6
 updates 22-6
Distributed queries 22-6
 identifying contributing CPUs 10-6
 LONG data in 6-6
Distributed updates 22-6
Distribution media 3-3
 files on 2-3
Distribution of data values 19-15
DML_LOCKS parameter D-14
DML locks 12-7
 see data locks
DML statements 1-5, 11-3
 against views 18-3
 example of executing 10-3
 in current uncommitted transaction 12-4
 privilege to perform 17-3
Documents
 storing 6-5
Drivers 9-9
 default single-task 9-9
 part of program interface 9-9
Driving index 19-10
Driving search criteria 19-11
Driving table
 choosing 19-15

DROP ROLLBACK SEGMENT 16-7
 syntax G-19
DROP statements 1-5
DROP TABLESPACE statement 3-6
 syntax G-20
DROPCAT5.SQL file A-3
Dropping database objects 1-5
Dropping indexes 4-12
Dropping tables 4-11
DTAB view (V5) A-3
DUAL view 7-5
Duplexing 15-27
Duplicate index values 10-6, 12-2
Duplicate values
 discarded by DISTINCT 19-5
Dynamic performance tables 7-10, B-6, E-29-33

E

EBCDIC data 6-2
ECHO
 setting B-37
ENQUEUE_HASH parameter D-14
ENQUEUE_LOCKS parameter D-14
ENQUEUE_RESOURCES parameter D-14
Enqueues
 displaying B-33
Error codes H-1
 changes to A-7
Error messages 15-3, H-1
Errors
 caught during parsing 10-6
 causing statement level rollbacks 11-8
 communicating 9-9
 communications mechanism 9-8
 during execution, causing rollback 11-8
 not caught by parsing 10-6
 parse, causing no rollback 11-8
Escalation, lock 12-14
EVENT parameter D-14
Exception handling 1-6
Exclusive locks 12-10, 12-12 - 12-13
Exclusive mode B-43
 database 14-3
Executable SQL statements 11-3

Execute phase 10-7, 12-3
Executing a SQL statement 10-4
Executing programs 8-2
Execution errors 11-8
Executions
 number to perform 10-7
EXIT command 11-3, B-31 - B-32
EXPLAIN.DOC file 19-11
Exponents 6-3
Export files
 difference from image backups 15-30
 editing 15-29
 Import options regarding 15-29
Export program
 exported data 15-30
 exporting a set of tables 15-30
 options of 15-29
 output file 15-30
Export/Import programs 1-7, 2-6, 15-29
Exports
 full database 15-30, 17-4
 privilege to use 17-3
Extents 4-1, 4-8, 4-14
 after first 4-10
 allocating 4-11, 20-9
 benefits of fewer 20-8
 current number of 20-8
 deallocating 4-11
 default values 4-10
 definition 4-9
 directory of 4-11
 failure to allocate a new 15-18
 initial 4-9 - 4-10
 INITIAL size 4-10
 list of current 4-11
 MAXEXTENTS limit 4-10
 maximum number of 4-10
 MINEXTENTS 4-10
 minimizing number of 20-8
 minimum number of 4-10
 NEXT size 4-10
 reduce allocations of 20-8
 rollback segment 15-18
 size of subsequent 4-10
 subsequent 4-9
 when multiple are preferred 20-8

F

Failure
 media 3-8
 in distributed databases 22-4
 instance 11-9
 operating system 11-4
 process 11-9
 program 11-3
 types of 15-3
Fetching results for a query 10-7, 10-8
Fields 1-4
File management locks 12-17
Files, database
 see database files
Files, operating system 1-11, 3-1
Files, ORACLE software 2-3
Filespecs 13-5, 16-8
FIXED_DATE parameter D-14
Fixed numbers
 storing 6-2
FLOAT datatype 6-7 - 6-8
Floating point numbers
 storing 6-2
FORCE option B-43
FOREIGN KEY clause A-8
Foreign keys 5-12
Format masks 6-4
Formatting features A-3
Fragmented databases 4-10
 allocating tables in 20-9
Free buffer waits statistic 20-14 - 20-15
Free buffers 20-11
 searching for 20-13
Free buffers inspected statistic 20-14, 20-15
Free list 5-4, 5-7
FREE_LIST_INST parameter D-14
FREE_LIST_PROC parameter D-15
Free space 4-11
 controlling 5-4
 data blocks 5-4
 effect of PCTUSED 5-7
FROM clause
 driving table in 19-16
Front compression, index A-6
Full database export 15-8

Full database import 15-31
 Full table scans 4-7, 19-6, 19-11 - 19-12, 19-16, 20-8
 against large tables 20-9
 effect of clusters 5-19
 vs. searching indexes 19-9
 when desirable 19-6
 Functions
 LONG data in 6-6

G

GC_DB_LOCKS parameter D-15
 GC_ROLLBACK_LOCKS parameter D-15
 GC_ROLLBACK_SEGMENTS parameter D-15
 GC_SAVE_ROLLACK_LOCKS parameter D-15
 GC_SEGMENTS parameter D-15
 GC_SORT_LOCKS parameter D-16
 GC_TABLESPACES parameter D-16
 GRANT CONNECT 17-2
 GRANT option 1-6, 5-26, 18-2
 privileges granted through 18-4
 GRANT RESOURCE statement 12-16, 16-9, 17-8, 18-2
 GRANT statements
 syntax G-21, G-23, G-25
 GRAPHIC datatype 6-7
 Graphics characters
 storing 6-6
 GROUP BY clause 4-18, 5-25, 19-5, 19-13
 LONG data in 6-6
 using WHERE 19-14
 Grouping tables 4-5

H

Hardware
 in network 22-7
 tuning 20-4
 Hashing
 accessing buffer cache 20-11
 HAVING clause
 vs. WHERE clause 19-14
 Head contention 4-5
 Head crash 15-4

HELP, online 3-5
 High activity data 4-5
 High update data 4-4
 Hit ratio B-23
 HOLD_CURSOR precompiler option 19-17
 Host database
 current 22-7
 initial 22-7
 Host machine 2-6
 Host name B-28
 setting B-37
 Host variables 10-3, 19-17
 binding values for 10-4
 Hour
 storing 6-4

I

IFILE argument to INIT.ORA D-16
 I/O 15-5, 20-7
 bottlenecks i 20-24
 combining writes 20-2
 DBWR too slow 20-24
 distributing across tablespaces 3-1, 4-4
 monitoring B-23, B-33
 optimizing 20-2, 20-6, 20-7
 primary concerns 20-6
 reading database files 20-6
 redo log files 20-17, 20-24
 using multiple files to control 20-7
 writing database files 20-6
 writing the redo log file 20-6
 I/O bound 20-23 - 20-24
 Image backups 15-10, 15-21
 Imbedded SQL
 array processing 19-17
 using SQL vs. host language 19-4
 IMMEDIATE mode B-41
 Import
 effect on passwords A-7
 full database 15-28
 Import program 15-31
 IN clause in queries 19-5
 Incremental exports 15-28, 15-29
 compressing files 15-31
 defined 15-31

Index data 5-4
accessing online with offline table data 4-6
changing tablespace for 20-10
format of 5-3, 5-15
index segments 4-12
location vs. table data 5-16
offline 5-16
overhead 5-15
separating from table data 4-4

Index extents
minimizing number of 20-8

INDEX privilege 5-14

Index segments 4-1, 4-8, 4-12, 5-2, 5-14
definition 4-12
reclaiming space 4-12

Indexes 3-3, 5-1, 5-11
altering storage 16-14
checking entries 5-18
column order 5-13, 5-14, 19-7
concatenated 19-16
controlling storage 4-2
created before data loaded 5-13
default storage parameters 4-3
dictionary locks on 12-16
driving 19-10
effect on SQL statements 5-11
entries 5-17
for offline tables 16-10
heavily accessed 20-10
internal loading 5-15
internal structure 5-17
invalid 5-18
length of columns in 16-13
LONG datatype and 6-6
maximum size 16-14
maximum storage 4-8
maximum width 16-14
merging multiple 19-10
multiple columns 5-13
names 5-14
non-unique 5-13
not null values 19-9
not used by DISTINCT 19-5
not used by GROUP BY 19-5
not used by NOT 19-12
null values and 19-9

offline 4-7, 16-10
optimal number of 5-15
performance impact 5-11
query using multiple 19-10
reclaiming space from 4-12
resolving query in 19-7
separating from tables 20-7
spanning files 4-5
storage for 5-11, 16-13
storing related tables 4-5
suppressing use of 4-7, 19-8, 19-10
 tablespaces and 4-4, 4-12, 16-13
unique 5-11, 19-16
unique vs. non unique 19-10, 19-15
updating 5-11
used by joins 19-14
used by optimizer 1-7
using effectively 19-1
using most selective 19-1
V6 compared to V5 A-6
validating 5-18
versus temporary segments 4-18
vs. keys 5-12
when not recommended 5-14
when recommended 5-13, 5-14, 19-11
when use is not possible 19-6
when use is possible 19-6

INDEXES view (V5) A-3

Infinity
internal representation 6-4

INITCAP SQL function F-7

INIT.ORA file 1-10, 8-1, 8-3, 13-3, 14-1 - 14-2, 14-5, 14-8, 20-4, A-4, D-1
for database creation 13-3
sample file 14-8
specifying alternate 13-3, 14-8, B-43
tuning for performance 20-1

INIT.ORA parameters 19-2, D-1

INIT_SQL_FILES parameter 7-8, 13-4, 13-6, D-16

INITIAL storage parameter 4-9 - 4-10, 5-2, 16-12, 16-13, A-6
recommendations for 20-8
to hold all data 20-9

Initialization
database A-2

INITTRANS option 5-4, A-6
INSERT statements 1-5, 6-5
 read consistency 12-8, 12-10
 remote 22-6
Inserting rows 4-9, 5-2
 performance 5-7
 prohibiting 5-6
 when allowed 5-4
Installation
 distributed databases 22-6
 drivers 9-9
 operating system privileges required 2-3
 ORACLE software 13-1
 other ORACLE products 13-7
 tuning during 20-5
Installing ORACLE 1-11, 2-2, 3-3, 9-7
 changes in software areas 8-2
Instance failure 3-10, 15-4
 causes of 15-7
Instance recovery 2-6, 3-9, 3-12, 4-7, 11-4, 14-5,
 15-3 - 15-4, 15-10 - 15-12, 15-15
 compared to media recovery 15-8
 impact of checkpoints and 3-12, 15-11
 in a shared disk system 15-4
 information for 15-9
 performed by SMON 9-8
 starting 15-7
Instance startup/shutdown 2-6, B-1, B-43
INSTANCES parameter D-16
Instances
 abnormal termination 4-15
 background processes associated with 9-6
 current 1-12, 13-4
 definition of 1-11, A-2
 names 1-12
 recovering 4-7
 rollback segment requirements 4-13
 sharing ORACLE code 8-2
 single-process systems 9-6
Instances, multiple
 database creation 13-2
 locks to control mounting 12-17
 writing to rollback segments 12-17
INSTR SQL function 19-4
INTEGER datatype 6-7 - 6-8
Integrity, data 12-2, 12-5

Interactive interfaces 1-8
Internal locks 12-17
INTERNAL user 15-15, A-3, B-3
 connecting as B-28
 for database creation 13-4
 for database maintenance 14-6, 16-3, 16-10,
 privileges required 14-4
 recommendations for use 2-5
Interrupt signal 15-26
INTERSECT 4-18
 LONG data in 6-6
INUSE rollback segments 20-21
IOR utility (V5) A-2, A-10

J

Joins 19-13
 clusters and 5-19, 5-20, 16-15
 data on multiple nodes 22-6
 driving table 19-15
 improving performance 5-14
 indexes for 5-11, 5-12
 non equi-joins 19-15
 on indexed columns 19-14
 on unindexed columns 19-14
 optimizing 19-14
 outer-equijoins 19-15
 subqueries transformed to 19-5
 unindexed 4-18
Julian dates 6-5

K

Kernel 1-7, 1-12, 8-2
 shared by instances 1-12
Keys
 generating unique 5-23, A-5
 vs. indexes 5-12
 sequences 5-23

L

LANGUAGE parameter D-16, F-2, F-7
 char_set argument F-2
 language argument F-2

setting on instance basis F-11
territory argument F-2
Large databases 3-1
Large redo writes 20-17
Latch-bound systems 20-23
Latches 12-6 - 12-7, 12-11, 12-17
 adjusting 20-23
 constraining performance 20-23
 descriptions of individual 12-17
 monitoring 12-18, 20-14, B-13, B-24 - B-25, B-33
Leading portion of index 5-16, 19-7
Leaf index blocks 5-15, 5-17
Least recently used algorithm
 for writing buffers to disk 15-5
Length
 column 5-9
LGWR background process 1-11, 3-8, 16-3,
 20-4
 optimizing writes of 20-11
 when woken 20-16
LGWR-bound system 20-23, 20-24
Loading clusters 5-20
Loading data 5-13
 checking for uniqueness 5-13
Local control (distributed databases) 22-8
Location
 binding by 10-7
Location transparency 22-2 - 22-3
 benefits of 22-4
LOCK TABLE statements 1-5, 12-6, 12-11,
12-12
Locking
 at parse time 10-4
 communications about 8-2
 information in SGA 8-3
 overriding default 12-8, 12-11
 to avoid deadlock 12-20
Locks 12-3
 automatic 12-5
 converting 12-14
 database privileges required 12-6
 definition of 12-5
 duration of 12-6
 escalating 12-14
 internal 12-7, 12-11, 12-13, 12-17
 monitoring 12-3, 12-18, B-11, B-33
obtained by DBWR 20-15
overriding default 12-5
released by PMON 9-8
released by ROLLBACK 11-8
table 12-13
V6 compared to V5 A-7
LOG files (spool) B-42
Log sequence numbers 3-9, 15-10, 15-19,
 15-24, B-35
 for manual archiving B-27
 listing B-26
 used during recovery 15-25
Log Writer (LGWR) process
 see LGWR Background Process
LOG_ALLOCATION parameter 15-13, 20-16,
D-18
 for shared disk systems 20-18
 for single instance systems 20-18
LOG_ARCHIVE_DEST parameter
 15-19 - 15-20, D-18
LOG_ARCHIVE_START parameter 15-19,
B-27, D-18
 default 15-21
LOG_BUFFER parameter 20-16 - 20-17, D-18
LOG_BUFFERS_DEBUG parameter D-19
LOG_CHECKPOINT_INTERVAL parameter
 15-12 - 15-13, 15-16, 20-6, 20-16 - 20-17, D-19
 increasing 20-17
LOG_DEBUG_MULTI_INSTANCE parameter
 D-19
LOG_ENTRY_PREBUILD_THRESHOLD
 parameter D-19
LOG_FILES parameter 8-4, 15-14, 16-3, D-20
 effect on control file 3-7
LOG_IO_SIZE parameter 20-16, D-20
Logging database changes 3-9
Logical structures
 vs physical structures 3-3
Logical unit of work
 see transaction
LOG_SIMULTANEOUS_COPIES parameter
 D-20
LOG_SMALL_ENTRY_MAX_SIZE parameter
 D-20
LONG datatype 5-14, 6-1, 6-5, 6-7
 column overhead 5-10

compared to LONG RAW 6-6
displaying B-38
maximum characters 6-5
restrictions on 6-6
versus LONG RAW 6-5
LONG RAW datatype 5-14, 6-1, 6-6
description of 6-6
versus LONG 6-5
LONG VARCHAR datatype 6-7
LONG VARGRAPHIC datatype 6-7
LONGWIDTH
setting B-37
Loops
PL/SQL 1-6
Lost updates 12-2
LOWER SQL function F-7
LRU list 20-11
DBWR cleaning 20-14
locked by DBWR 20-15

M

Mandatory columns 6-7
Manual archiving 15-10, 15-19, 15-21, 16-3
Master-detail tables 5-19
MAX function 19-16
MAXDATAFILES argument 3-4, 13-3
MAX_DUMP_FILE_SIZE parameter D-20
MAXEXTENTS storage parameter 4-10, 16-12
for rollback segments 4-16
MAXINSTANCES argument 13-3
MAXLOGFILES argument 3-11, 13-3, 15-14, 16-3
MAXTRANS option 5-4, A-6
Media failure 3-8 - 3-10, 15-4, 15-8
online redo log files 15-27
recovery requirements 3-9
Media recovery 2-6, 3-9, 14-2 - 14-3, 15-3, 15-21, 15-24
compared to instance recovery 15-8
interrupting 15-26
minimizing time required for 15-22
possibility with NOARCHIVELOG mode 15-11
requirements for 15-8

Memory
increasing requirements on 20-12
optimizing 20-2
process limits on 8-7
required 8-1, 20-5
Menu driven products 1-8
MESSAGES parameter D-21
Messages
between processes and memory 8-2
language files for F-2, F-3
MIN function 19-16
MINEXTENTS storage parameter 4-10, 5-2, 16-12, 16-13, A-6
MINUS 4-18
LONG data in 6-6
Minute
storing 6-4
Mode, redo log
altering 15-9, 15-14
choosing 15-13
default 15-14
Modes, lock 12-13
MONITOR command 12-3, 20-1, B-33, B-38
directing output to a file B-6
LATCHES option 20-14, 20-23
STATISTICS option 20-12, 20-14, 20-17
ROLLBACKoption 16-6
MONITOR.SQL file B-6, E-29
Monitoring database use 1-7, 2-6, 14-4, B-1
Months
names in local language F-2, F-4
representation 6-4
Mounted databases 14-3
Mounting a database 14-5, B-1, B-43
shared or exclusive mode 12-17
MS-DOS operating system
single-process systems 9-6
Multi-user server databases 22-5
Multiple instances
sharing a database B-43
Multiple-process systems 9-2, 9-6
Multiprocessor machines 20-2
MYPRIVS view 7-5

N

National language support A-8, F-1
problems associated with F-6
requirements and purpose F-2
supporting files F-2
Negative numbers 6-3 - 6-4
Nested queries
 LONG data in 6-6
Network protocol
 in DBlink 22-9
Networking
 DBA issues related to 22-6
 role in distributed database 22-3
 software 1-12, 22-7
NEXT storage parameter 4-10, 16-12, A-6
 setting sizes for 20-9
NEXTVAL 5-24 - 5-25
 used with CURRVAL 5-25
NLS_SORT parameter D-21, F-2
NLSSORT function A-8
NOARCHIVELOG mode 3-8 - 3-9, 14-3,
 15-9 - 15-10, 15-13, 15-18, 16-2
 implications of using 15-11
 switching to 15-15, 15-16
 tablespace failure and 15-11
 vs. ARCHIVELOG mode 3-9
NOAUDIT statements 12-16, 17-13, 18-8
 syntax G-27, G-28
NOCOMPRESS index option 5-15
Nodes
 client 22-5
 in DBlink 22-9
 linking distributed 22-5
 multiple in one SQL statement 22-4
 partitioning data among 22-3
 querying data on several 22-6
 server 22-5
Non-shared code 8-2
NORMAL mode B-41
Normal termination 11-3
Normalization 19-2, 20-3 - 20-4
IS NULL clause 19-9
NOT NULL clause 6-7
NOT NULL data
 selecting 19-6

NULL clause A-8
Null data 1-4
 allowing 5-2
 definition of 6-7
 in cluster index 5-20
 in indexes 5-19, 6-7, 16-13
 selecting 19-6
 selecting all but 5-14, 19-6
 sort order A-8
 storing 6-7, A-6
 versus zero 6-7
 when stored 5-9
NUMBER datatype 6-1, 6-7 - 6-8
 description of 6-2
 displaying B-38
 internal format 6-3
 maximum value 6-2
NUMWIDTH
 setting AB-37

O

Objects, database
 auditing of 18-5
OCI (ORACLE call interface) 9-9
ODL utility (V5)
 superceded by SQL*Loader A-3
ODS utility (V5) A-2
Offline data 3-6
 SQL statements referencing 4-6
Offline index
 online table 5-16
Offline redo log files 3-10, 15-8,
 accessing 15-10
 applying 15-6
 inventorying 15-19
 restoring 15-25
 where written 15-19
Offline table
 online index 5-16
Offline tablespaces 3-6, 4-17, 14-7
 bringing online 16-10
 reasons for 16-9
Online backup 15-3
Online data
 dependencies on offline data 4-6

Online help 3-5, 4-3
Online redo log files 3-10, 15-9 - 15-10
 archiving 3-10, 15-10, 15-12
 archiving with ARCH 9-8
 backing up 15-22
 changing configuration of 15-15
 checkpoints 3-12, 15-11
 checkpoint interval 15-12, 15-16
 creating 15-13, 15-14
 default 15-14
 device allocation 15-13
 displaying current 15-19
 dropping 15-15
 enlarging 16-3
 LGWR writing to 9-7
 media failure in 15-8
 number of 15-13, 15-14
 on different device than database files 15-15
 preparing for reuse 3-12, 15-11
 renaming 15-15
 reusing 15-10, 15-12
 size of 15-13, 15-15
 when all full 15-18
 writing 15-10, 15-23
Online tablespaces 14-7
Open databases 14-3
OPEN_CURSORS parameter 8-4 - 8-5, 8-7,
 10-5, D-21
OPEN_LINKS parameter 8-4, D-21
Opening a database 14-4 - 14-5, B-1, B-43
Opening databases 1-10
 by multiple instances 1-11
Operating system failure 15-4, 15-8
Operating system files 1-10
Operating systems
 moving data between 1-7
OPI (ORACLE program interface) 9-9
OPS\$ login usernames
 creating 17-7
 passwords for 17-7
Optimizer 4-7
 definition of 1-7
 evaluating indexes 19-6
 rules of 19-11
OR clauses
 indexes and 19-12

suppress index use 19-13
ORACLE RDBMS V6 with and without
 transaction processing option 1-3
ORACLE software
 installing other 3-5
ORDER BY clause 4-18, 5-25, 19-13
 access path 19-16
 LONG data in 6-6
 non-default character set F-10
Order of index columns 5-15
Ordering results 10-8, 19-13
Outer joins 19-15
Overhead
 data block 5-3

P

Paging 20-12
 effect of SGA on 20-12
 of SGA 8-3
Parameter file
 see INIT.ORA file
Parameters, INIT.ORA 14-1, 14-8, A-4
 changing 14-6
 tuning for performance 14-9
Parentheses
 to show precedence 19-3
Parse locks 12-7, 12-16
 acquiring 10-6
Parsing 10-2 - 10-3, 10-6, 19-17
 at server node 22-5
 DDL statements 10-8
 definition of 10-3
 locks held during 12-16
 locks on dictionary caches 12-17
 locks released 12-17
 reducing frequency of 10-5
Partitions(V5) A-8, A-9
 equivalent to tablespaces AA-5
Passwords B-28
 changing 17-9
 encrypted 17-3, A-7
 operating system 17-2
 ORACLE 17-2
Paths, access

optimization 19-11
ranking 19-15
using ROWID 19-11
Pattern matching 19-11
PCTFREE 4-10, 5-4, 16-12, A-6
 changing 16-13
 clusters 5-22
 default and range 5-5
 examples 5-7
 in clustered block 5-5
 in non-clustered block 5-5
 recommendations for 5-5
 restrictions on setting 5-7
 setting 5-6
 when to increase 20-9
PCTINCREASE storage parameter A-6
PCTUSED 4-10, 5-4, 5-6, 16-12, A-6
 adjusting with PCTFREE 20-9
 changing 16-13
 clusters 5-22
 default and range 5-6
 examples 5-7
 restrictions on setting 5-7
 when to use non-zero 20-9
Performance 8-2
 affect of background processes 9-6
 distributed databases 22-6
 effect of INIT.ORA on 14-8
 impact of archiving 15-9
 impact of storage on 16-1
 measuring 20-3
Performance tuning 3-3, 12-2
 impact of TABLE_LOCKING and
 SERIALIZABLE 12-20
 quick steps for 20-6
 tuning DBWR process 20-13
Peripheral memory 8-2
Physical reads 20-7
 statistic 20-12
Physical structures 3-3
 vs logical structures 3-3
Physical writes 20-7
 batching factor 20-16
 statistic 20-12
"Piggy-backed" commits 20-16
PL/SQL 1-3, 1-6, A-2
PMON background process 1-11, 9-8, 15-4
Ports
 software version numbers 1-8
Positive numbers 6-3
Power failure 15-4, 15-7
Precision 5-2, 6-2
 using 6-3
Predicates 19-6
 and index use 19-10
 evaluating 19-8
 merging multiple 19-10
PRE_PAGE_SGA parameter D-21
PRIMARY KEY clause A-8
Primary keys 5-11, 5-19
Priorities
 background processes 20-5
 user processes 17-3, 20-5
Private DBlinks 22-10
Private rollback segments 4-16
 creating 16-6
Privileged SQL*DBA commands 13-3, 14-4
Privileges
 on tables 18-2
 required to lock 12-6
 revoking 18-4
 SQL*DBA 14-4
Privileges, account 2-3, 20-5
Procedures 1-6
PROCESSES parameter D-22
Process
 definition of 9-2
 memory allocated for 9-2
Process failure 15-4
Process global area
 see Program Global Area (PGA)
Process Monitor (PMON) process
 see PMON background process
Process recovery 9-8
Processes, user
 communication between 8-2
 control information for 8-4
 displaying B-33
 failure in user 9-8
 memory and number of cursors 10-5
 monitoring 8-4, B-8
 running out of memory 8-7

separating from ORACLE programs 9-8
types of 9-2
Program (ORACLE RDBMS) files 8-2
Program Global Area (PGA) 8-1, 8-4
size of 8-4
Program interface 9-8
in two task systems 9-4
single task systems 9-3
Programs, user-written
currently in use B-5
installing shared 8-2
setting initial cursor size 8-5
Protocols 22-2
in DBlink 22-9
options in DBlink 22-9
Protocols, communications 9-9
Pseudo-columns 5-24
Public DBlinks 22-10
advantages of 22-11
Public rollback segments 4-16
creating 16-5
PUBLIC synonyms 17-14
privilege to create 17-4
same name as tables 17-14
PUBLIC user 17-14

Q

Queries 1-5
data locks allowing 12-13
data seen by 12-3
different from other SQL statements 10-7
distributed 22-8
executing 10-3, 10-8
implicit 10-8
improving performance through clusters
16-15
issues relating to executing 10-8
many concurrent 20-22
numerous short 20-22
optimizing 1-7, 19-11
ordering output 19-13
privilege to 17-3
read consistency 12-8, 20-21
reconstructing data for read consistency 12-3
resolving using index only 19-7

single row paths 19-5
single table, nested 16-15
using ROWID 6-8
work area for 4-18
Query paths 19-5
ranks of various 19-15
using indexes 19-6
Query performance 5-11, 5-13, 5-20
vs. update performance 5-15
Quotas 5-2, 17-4
changing 16-9, 17-6
storage 16-9, 16-11
user 4-2, 4-4

R

RAW datatype 6-1, 6-6
description of 6-6
Read consistency 4-12, 4-15, 10-8, 12-3, 15-5,
15-30, 16-5, 20-21
effect of PCTREE 5-6
examples of 12-4
for a series of statements A-7
information in rollback segments 15-17
information not available 4-15
multiple versions 20-21
Read only access 18-2
Read only data 4-4
Read only transactions 11-4, 12-15
Reading the database 20-10
Real memory 8-2
Rear compression, index A-6
REBIND precompiler option 19-17
Reclaiming space 4-11
RECOVER command B-35
RECOVER DATABASE command 15-26
RECOVER TABLESPACE command 15-25
Recovery 1-7, 4-7, 4-12, 14-4, 20-10
DBA decisions relating to 15-13
impact of checkpoints on time required for
15-12
tablespace 4-3
to past point in time 15-27
see also Instance recovery
see also Media recovery
Recovery data 1-10

Recovery time 15-3
RECOVERY.DOC online file B-35, 15-29
Recursive calls 8-5
Redo Log 3-8, 4-19
 and rollback segment data 3-9
 choosing mode 3-11, 15-9
 contents 3-8
 how written 3-8
 loss of 3-8
 media failure in 15-27
 modes 3-9, 16-2
 monitoring 20-17
 optimizing 20-16
Redo log buffers 20-16
 notifies LGWR when full 20-16
 size 20-17
 writing to redo log 15-11, 20-16
 written at commit 20-16
 written by DBWR 20-17
Redo log files 1-10, 15-6, 16-1
 adding 16-3
 archiving 3-9, B-26
 at startup 14-5
 automatic or manual creation 13-2
 block size 3-11
 blocks to be committed 15-6
 changing size of 16-3
 compared with AIJ A-4
 contents 3-9
 creation 13-2, 13-6
 directly accessing 3-9
 dropping 16-4
 listing range of B-26
 maintenance on 14-3, 14-6, 16-2
 named in control file 3-7
 number of 3-11
 on tape 3-10
 optimal placement 20-6, 20-11
 recovering 15-4
 renaming 16-4
 requirements 3-8
 reusing 3-9
 rollback data in 4-14
 setting instance allocations 20-18
 size of 3-11
 writing to 20-11

see also Online redo log files
Redo log management locks 12-17
Redundant data 22-8
Referential constraints 5-2
Referential integrity A-8
Relational database management systems 1-2
Relational theory 19-2
Release media 3-3
Release number, ORACLE 1-8
Releasing locks 12-5
Releasing resources 15-7
REM command B-36
Remarks
 in command files B-36
Remote databases
 connecting to 22-2
 using SQL*DBA on 14-4
Remote instances
 starting B-37
 starting 14-4
Renaming files
 redo log files 16-4
Reparsing 19-17
Repeatable reads 12-11, 12-21
REPLACE function A-8, F-9
Reserved words 10-2, A-8
 for datatypes 6-7
RESOURCE privilege 5-2, 5-24, 16-11, 16-13,
 17-4
 and auditing 18-5
 on tablespaces or across system 17-4
 quotas 16-9
 tablespaces 4-4
 with or without quota 17-6
Resources, database
 freed by PMON 9-8
 holding exclusively 12-12
 required to execute a statement 10-3
 sharing 12-12
 to be locked 12-5
Response time 22-7
Retrieving data 1-5
REUSE option
 for existing database files 13-3, 13-5
Revision level, ORACLE 1-8
REVOKE statement 17-6, 17-9, 18-2, 18-4

syntax G-29, G-30, G-31
Rollback data 4-15, 4-14
 deferred 4-17
 when released 15-17
Rollback recovery 14-5
Rollback segment locks 12-17
Rollback segments 3-5, 4-1, 4-3, 4-8, 4-12, 15-6,
 20-4, A-5
 active 3-6
 adding new 15-16
 allocating extents 4-14, 15-18
 allocating PUBLIC 4-16
 applying rollbacks using 15-6
 assigned to instances 20-22
 assigned to transactions 15-18, 20-20
 caching 20-22
 changing storage for 16-6
 contents 4-12, 4-14
 creating 4-10, 16-5
 data 3-9
 deallocating 4-14
 deferred 4-17
 definition 4-12
 dropping 16-6
 filled by one transaction 20-21
 information in data dictionary 16-6
 locking segment header 20-21
 maintenance on 16-1
 making private 4-17
 MAXEXTENTS for 4-16
 multiple 4-13, 16-5, 20-6, 20-21
 number of 15-13, 15-17
 optimal size 20-22
 private 15-13
 private vs. public 4-16, 16-5
 public 15-13
 required number 4-16
 size of 15-13
 space required by 15-18
 status of 20-21
 SYSTEM 4-13, 13-6, 15-16, 16-5
 tablespace allocation 15-13, 16-8
 transactions per 4-14, 15-17
 tuning 20-20
 used for read consistency 12-3, 20-22
rollback statistic 20-21

ROLLBACK WORK statement 1-5, 11-3, 11-5, A-7
 implicit and explicit 11-9
 results of 11-8
ROLLBACK_SEGMENTS parameter 4-17, 16-5, 20-21, D-22
Rolling back 4-12, 4-14, 4-15, 16-5, 20-20
 aborted processes 15-4
 at shutdown 14-10
 at startup 14-5
 data for 4-14
 deadlock detection 12-19
 effect on sequence numbers 5-23
 implicit A-7
 normal transaction 15-5
 SQL statement 11-8, 15-3
 to savepoint 12-10, AA-7
Rolling forward 3-8, 14-5, 15-5 - 15-6, B-35
 to specified time 15-6
 with NOARCHIVELOG mode 15-16
Rounding of numeric data 6-3
 forcing via scale 6-3
Row address
 see ROWIDs
ROW_CACHE_BUFFER_SIZE parameter D-22
ROW_CACHE_ENQUEUES parameter D-22
ROW_CACHE_INSTANCE_LOCKS
 parameter D-22
ROW_CACHE_MULTI_INSTANCE
 parameter D-22
ROW_CACHE_PCT_FREE parameter D-22
Row data
 fetching 10-8
 locking with data locks 12-7
Row directory 5-3
Row exclusive locks 12-10, 12-13
Row headers 5-9
Row length
 average 5-7
 vs ORACLE block size 3-5
Row level lock manager 1-3, 12-21, 20-2, A-7
ROW_LOCKING parameter D-23, 12-8, 12-20
Row locks 12-10
 (TA locks) 12-18
Row pieces 5-9, 6-9
 addresses of 5-3

chained by ROWID 5-9
 effect of chained on performance 5-5
 information in row header 5-9
 inserts or updates causing multiple 5-9
 reducing chaining 5-6
 Row sequence number
 in ROWID 6-8
 Row share locks 12-13
 ROWIDs 5-10
 chaining row pieces 5-9
 components of 6-8
 description of 6-8
 in indexes 5-15, 5-17
 in query path 19-5, 19-11
 uses for 6-9
 V6 compared to V5 A-6
 versus columns 6-9
 Rows 1-4
 limiting access to certain 18-2
 accessing via indexes 5-11
 accessing via ROWID 6-9
 array processing 19-17
 concurrent access 20-13
 data spanning blocks 5-4
 expected length 16-12
 identifying via ROWID 6-9
 number per table 19-15
 overhead required 5-10
 per cluster block 5-22
 unique 5-11, 5-12
 Rows per block
 decreasing 5-6
 increasing 5-5
 one 5-5, 20-9
 when fewer is better 20-8
 when to reduce ratio 20-13
 RS locks 12-13
 RX locks 12-13

S

S locks 12-13
 SAVEPOINT statement 11-5 - 11-6, A-7
 in ORACLE tools 11-9
 Savepoints 4-15
 implicit 11-8
 locks released on rollback to 12-10
 maximum active 11-7
 naming 11-6
 SAVEPOINTS parameter 8-4, 11-7, D-23
 Saving partial transactions 11-6
 Scale 5-2, 6-2
 using 6-3
 SCN_INCREMENT parameter D-23
 SCN_THRESHOLD parameter D-23
 Second (time)
 storing 6-4
 Secondary storage 8-2
 Security, database 2-2
 Segment header block 4-11
 contention on 20-21
 Segments 1-11, 4-8
 allocating extents 4-11
 index 5-11, 5-14
 spanning files 4-8
 SELECT FOR UPDATE statement 12-6, 12-11
 when recommended 12-14
 SELECT lists 6-9
 using LONGs in 6-5
 SELECT statements 1-5, 10-8
 read consistency 12-8
 Selection criteria
 evaluating 19-8
 Self joins 5-21
 SEQUENCE_CACHE_ENTRIES parameter
 5-26, D-23
 SEQUENCE_CACHE_HASH_BUCKETS
 parameter D-24
 Sequences 5-1, A-5, A-8
 ascending or descending 5-23
 altering definition of 5-26
 auditing 5-26
 creating 5-24
 dropping 5-26
 exporting/importing 5-26
 generating with NEXTVAL 5-24
 number cached 5-26
 privileges on 5-26
 reusing 5-23
 skipped 5-25
 SERIALIZABLE parameter 12-8, 12-20, D-24
 effect of setting to TRUE 12-21

Server database 22-5
 specified in DBlink 22-9
Server processes 9-2
Session numbers
 using sequences to identify 5-26
SESSIONS parameter D-24
Sessions
 auditing by 18-4
 using sequence numbers in 5-25
SET clause (UPDATE statement) 6-5
SET command B-37
SET INSTANCE command 22-7
SET TRANSACTION READ ONLY statement
 12-7, A-7
SGA
 see System Global Area (SGA):
Shadow processes 9-2, 9-4
Share locks 12-13
Share row exclusive locks 12-13
Share update locks 12-13
Shared code 20-5
 user programs 8-2
Shared disk systems 1-12, 4-13, 4-16, 15-16
 backup strategy 15-2
 file and log management locks 12-17
 mounting shared mode 14-3
 rollback segments required 4-13, 16-5
 settings for TABLE_LOCKING,
 SERIALIZABLE 12-20
Shared Global Area
 see System Global Area (SGA)
Shared locks 12-12
Shared memory 9-6
Shared mode (database) 14-3, B-43
Sharing data 2-3
SHOW command B-39
SHOW SGA command 8-3
SHUTDOWN command 14-3 - 14-4, 14-6,
 14-10, 15-7, B-3, B-41
 ABORT option 14-11
 IMMEDIATE option 14-10
 NORMAL option 14-10
 offline tablespaces 4-6
Shutting an instance 1-7, 1-10, B-41
Sign (numeric data) 6-3
Single row query paths 19-5

Single task systems 9-3
Single-process systems 9-6, 22-5
SINGLE_PROCESS parameter 9-6, D-24
Site autonomy 22-2, 22-4
 benefits of 22-4
SIZE cluster option 5-22, 16-15
 default 5-22
Small redo writes statistic 20-16 - 20-17
SMALLINT datatype 6-7 - 6-8
SMON background process 1-11
Snapshot too old 4-15
Software
 choosing optimal node for 22-7
 tables required by ORACLE 4-3
Software areas 8-1 - 8-2
Software failure 3-9
Software requirements
 backup 15-2
Software versions 1-8
 moving between 15-29
Sort/merge join 19-13, 19-14, 19-16
 INIT.ORA parameters 19-13
SORT_AREA_SIZE parameter 19-13, D-24
Sorting 4-18, 8-5
 based on non-English character set F-2, F-10
 ordering of nulls A-8
SORT_READ_FAC parameter D-25
SORT_SPACEMAP_SIZE parameter D-25
Space definitions (V5) A-5, A-8
SPOOL command 12-18, B-42
Spreadsheet applications 1-8
SQL.BSQ file 7-8
SQL data language 1-2 - 1-3
 as implemented by Oracle Corp. 1-3
 automatic optimizations 19-5
 categories of 1-5
 changes in V6 A-8
 standards A-3
SQL files B-2
 nesting B-4
 running B-4
SQL interface
 part of program interface 9-9
SQL statements 1-3, B-2
 against offline tables 4-7
 auditing select 18-4

checking construction of 10-6
context area for 8-5
current state described in cursor 10-2
definition 10-2
executable 11-3
executing B-3
executing from SQL*DBA 14-4
failure in execution 11-8, 15-3
good practices 19-3
interpretation of 1-7
loading into a cursor 10-3
multiple cursors 10-4
number per transaction 11-2
parse locks on objects referenced in 12-16
processing 4-18, 10-1
re-executing 10-4
referencing multiple nodes 22-4
referencing offline data 4-6
reparsing 19-17
replaced in cursors 10-6
sent through a network 22-5
storing text of 8-5
successful execution vs. commit 11-2
terminating B-36
translated form of 8-5
translating via parsing 10-3
versus SQL*DBA commands 1-4
waiting to execute 12-2
wording of 19-1, 20-4
writing to suppress index use 19-11

SQL*Connect 22-2
SQL*DBA 1-10
 using across CPUs B-4
SQL*DBA commands 1-3, B-2
 described B-2
 privileged 13-3, 14-4
 running in batch B-4
 versus SQL statements 1-4
 who can use B-3

SQL*DBA program 1-7, 2-6, 14-1
 access to 14-4
 database creation 13-4
 exiting B-31
 features of A-2
 invoking B-2
 primary uses of 14-4

privileged commands 2-3, 2-5
setting session characteristics B-37
uses of B-1
 using for recovery 15-8
 using MONITOR 12-18

SQL*Loader 1-7, 2-6
 supercedes ODL A-3

SQL*Net 1-13, 22-2, B-4
 program interface and 9-9
 two task structure 9-4

SQL*Plus
 compared to SQL*DBA A-3
 COPY command 22-8

SQL*Star
 definition of 22-2

SQL/DS databases 22-2
SQL/DS datatypes 6-7 - 6-8
SRX locks 12-13

Starting a database
 see Starting an Instance

Starting an instance 1-7, 1-10, 2-6, 3-6, 8-3,
14-4 - 14-5, B-1, B-43
 access to rollback segment 4-12
 allocating SGA 8-3
 archive options in effect 15-20
 automatic 14-7
 background processes 9-7
 control file 3-6
 failure during 15-28
 files required 3-4
 INIT.ORA parameters required for 14-8
 number of database files 3-4
 number of log files 3-11
 recovery during 9-8, 15-3 - 15-4, 15-7
 remote B-4, B-37
 single-process or multiple-process 9-6

STARTUP command 14-3 - 14-5, 15-7, B-3,
B-43

Statement failure 15-3

Statement level rollback 4-15, 11-8, A-7 - A-8

Statistics 20-1
 monitoring B-15, B-33
 system B-5

STOPONERROR option B-37

Stopping a database
 see Shutting an instance

Stopping an instance
see Shutting an instance
Storage
 index vs. table data 5-16
 limiting via quotas 4-2
 monitoring 16-1
 table vs. cluster 5-2
STORAGE clause A-6, G-33
Storage devices 20-5
 allocating files across 20-10
Storage parameters 4-1, 4-5, 4-8, 5-2, 20-7, A-6
 adjusting for performance 20-8
 altering 16-9, 16-13
 clusters 16-15
 default tablespace 4-3
 in effect 4-11, 16-11
 index 5-14
 overriding 4-3
Storage requirements (overall) 20-5
Stored queries (views) 5-10
Striping tables 20-8, 20-10
Structures, database
 changes to 12-5
 locking 12-7
Subqueries
 transformed to joins 19-5
SUBSTR SQL function 6-9, 19-4
Synonyms
 on remote tables 22-9
 privilege to create 17-3
 translating into base tables 10-6
SYNONYMS view (V5) A-3
Syntax, SQL
 effect on performance 20-4
 errors 11-8
SYS user (DBA) 2-3 , 17-4 - 17-5
 altering tables owned by 2-4
 objects belonging to 2-4
 password 2-4
 versus INTERNAL 2-5, B-28
System global area (SGA) 1-11, 3-5, 4-15, 8-1,
 8-3 - 8-4
 allocating 8-3, 14-2, 14-5
 buffer cache 20-11
 contents 8-3
 effect of increasing buffers 20-12
 in real versus shared memory 8-3
INIT.ORA parameters affecting 14-8
 latches on 12-17, 20-23
 locking 12-5
 optimizing 20-7
 size 8-3, 14-6
 use of real memory 8-3
 used to access offline data 4-7
System Monitor (SMON) process 9-8
System objects 12-5
SYSTEM partition (V5) A-4
System privileged SQL*DBA commands 14-4,
 17-5, A-2, B-3
SYSTEM rollback segment 4-13, 20-6
 and SYSTEM tablespace 16-5
 used by RDBMS and background processes
 20-22
SYSTEM tablespace 3-4, 3-6, 4-2 , 4-18, 16-17
 and SYSTEM rollback segment 16-5
 bootstrap segment 4-19
 data dictionary 4-3
 enlarging 4-3
 online requirement 4-3, 4-6
 original files 13-6
 recovering 15-4
SYSTEM user (DBA) 2-3 - 2-4, 17-5
 objects owned by 2-4
 password 2-5

T

TA locks 12-18
TAB view (V5) A-3
Table creation
 desirability of NOT NULL 19-9
 failure due to lock on cluster 12-16
Table data 5-2, 5-4
 access privileges 18-2
 accessing 20-8
 accessing online with offline index 4-6
 consistent view of multiple tables 12-11
 consolidating fragmented 20-8
 data segment 4-12
 deleting all 4-11
 format of data block 5-3
 guaranteeing unique rows 5-12

when to use many extents 20-8
Table definition
 dictionary lock on 12-16
 protecting 12-2
Table directory 5-3
Table locking 12-21
Table names 1-4, 5-2
 prefixed by OPS\$ 17-7
 same as public synonyms 17-14
Table privileges 12-6
TABLE_PRIVILEGES 7-5, E-18
Tables 1-4, 5-1 - 5-2
 auditing 18-8
 belonging to dropped users 17-9
 clustering 5-19 - 5-21, 16-15 - 16-16
 column order 5-9
 controlling storage 4-2
 creating 4-9, 6-7
 creating non-clustered 12-16
 default auditing for 18-6
 default storage parameters 4-3
 default tablespace 5-2
 definition of base 19-2
 definitions validated during parsing 10-6
 dropping clustered or unclustered 16-17
 exporting 15-29
 heavily accessed 20-10
 IDs in MONITOR LOCKS 12-18
 indexing 5-14
 listing in FROM clause 19-17
 locking 12-17, 12-12
 LONG columns per 6-6
 offline 5-16, 16-10
 offline indexes for 16-10
 overhead storage 6-9
 primary keys 5-11
 querying large 19-11
 required by ORACLE software 4-3
 revoking privileges 18-4
 separating from indexes 4-4, 5-16, 20-7
 spanning files 3-3, 4-5
 spreading across files or tablespaces 20-10
 storage 4-5, 4-8, 4-9, 16-1, 16-11, 16-12
 storing related indexes 4-5
 striping large 20-9, 20-10
 tablespace in 4-4, 4-5, 16-11, 20-10

Tablespace failure 3-10
 under NOARCHIVELOG mode 15-11
Tablespace locks 12-17
Tablespace recovery 15-3
Tablespaces 1-11, 3-3, 4-1 - 4-2, 16-7
 adding files to 16-8
 adding new 3-5, 4-7, 16-17
 allocating data to 20-10
 and database files 16-2
 backing up 4-6, 15-22
 bringing online 4-2, 4-6, 4-17, 15-26, 16-10
 changing allocation of data 20-10
 cluster index 5-20
 compared to partitions A-5
 creating 4-7, 16-8
 creating objects in 4-4
 dedicated to one application 20-7
 default 4-4, 5-2, 16-11, 17-5, 20-7
 default for table creation 16-11
 default storage parameters 16-7, 16-9
 dropping 16-10
 enlarging 4-7, 16-8
 failure in non-SYSTEM 15-24
 first one after SYSTEM 16-8
 indexes 5-14, 5-16
 information in data dictionary 16-8
 maintenance of 16-1, 16-7
 monitoring storage for 16-8
 multiple 15-16, 16-5
 offline 3-6, 3-10, 4-6, 15-24, 16-10
 online 3-4, 16-8, 16-10
 online at startup 14-7
 online or offline 12-17
 online requirement 4-3
 PUBLIC access to 17-8
 purpose of 4-2
 quotas 16-9
 recovering 15-4, 15-8, 15-24, B-35
 renaming files in 16-9
 RESOURCE privilege on 4-4
 revoking access 17-6
 rollback segments required 4-13
 segregating index and table data 5-16, 20-7
 storage characteristics 4-3
 SYSTEM 4-4
 taking offline 4-2, 4-4, 15-25, 16-9

temporary segments 16-18
using just SYSTEM 4-4
using multiple 4-4
with active rollback segments 3-6

Tablespaces, offline 4-17
accessing data in 4-7
automatic 4-6
bringing online 4-6
cautions about 4-6
recovering 4-7
use of 4-6

Tape drive
dedicated for archiving 9-8

Tasks
see Processes

TD locks 12-18

Temporary data 4-4

Temporary segments 4-1, 4-3, 4-8, 10-8, 16-17, 20-3
allocation of 4-18
and quota 17-6
at instance startup 16-17
cleaned up by SMON 9-8
data dictionary information 16-17
default 17-5, 17-8
default tablespace for 4-4, 16-17, 20-7
monitoring 16-1, 16-17
storage parameters 4-18, 16-18
 tablespaces for 4-18
TT locks on 12-18
used by sort/merge 19-13
V6 compared to V5 A-6
versus indexes 4-18
when dropped 16-17
when required 4-18

Terminals B-2
names of B-2
redirecting output B-37

TERMOOUT
setting B-37

Territory argument to LANGUAGE F-2

Time
24 hour format 6-4
default 6-4
entering 6-5
ignoring for date comparisons 6-5

TIMED_STATISTICS parameter B-25, D-25

Timeouts
DBWR 20-14
frequent 20-23
in MONITOR LATCHES display 20-14

TIMING
CPU and elapsed B-37
setting B-37

TM locks 12-18

TO_CHAR function 6-5, F-4

TO_DATE function 6-4, F-4

TO_NUMBER function 6-5

Trace files 15-7
written by ARCH 15-19

Trailing nulls 5-9

Transaction IDs 4-14

Transaction locks 11-9, 12-18
transaction processing option 1-1
locking characteristics of 12-8

TRANSACTIONS parameter D-25

Transactions
abnormal termination 15-5
active 4-15, 9-8
affecting concurrency 12-3
and rollback segments 4-13
average duration 20-3
average number per rollback segment 15-17
committing at disconnect B-30
committing upon exit from SQL*DBA B-31
committing work of 11-5
data consistency throughout 12-8
deadlock detected in 12-19
deferred rollback of 4-17
definition of 11-2
defining 11-4
duration of locks 12-6
ending 11-3, 11-8
entries 5-4
failure to COMMIT or ROLLBACK 11-3
IDs in MONITOR LOCKS 12-18
in ORACLE tools 11-4
interrupted at shutdown 14-10
last one before disconnect 11-5
long 4-15, 20-17
maximum number of concurrent 15-17
normal termination 15-5

number of SQL statements 11-2
per rollback segment 4-14
per write - batching factor 20-16
read only 11-4
repeatable reads 12-11
rollback entries for 15-18
rolling back 4-12, 15-5
rolling forward 15-5
savepoints during long 11-6
serializing 12-21
sharing rollback segments 4-14
short 4-15
starting 11-3
types of 11-2
writing to rollback segment 4-14

TRANSACTIONS_PER_ROLLBACK_SEGMENT parameter 20-21, D-25

Transformations
SQL statements 19-5

Translations
character set problems F-6

TT locks 12-18

Tuning a database 20-1

Tuples 1-4

Two-task systems 9-2, 9-4

U

UID (username and password) 22-10
DBlink 22-9

Unbounded range query path 19-11, 19-16

Uncommitted data 15-5, 15-7

UNION clause 4-18
LONG data in 6-6
vs. OR clauses 19-12

UNIQUE indexes 5-12

Unique values 5-11, 5-14, 16-13
loading 5-13

Update performance 5-6 - 5-7
vs. query performance 5-15

UPDATE statements 1-5, 6-5
read consistency 12-8, 12-10

Updates
based on another table's data 12-15
data locks allowing 12-13
effect on free space 5-4

effect on indexes 5-15
remote 22-6
requiring more space than original 5-4
timing of 12-2
when allowed 5-4

Upgrading ORACLE software 1-11, 8-2, A-1

UPI (User Program Interface) 9-9

UPPER SQL function F-7

USE_ROW_ENQUEUES parameter D-26

USER data dictionary views 7-4

User objects 12-5

User process
failure in 15-4
separate from ORACLE program 9-4

User processes 9-2 - 9-3
shadow processes for 9-2

USER_AUDIT_CONNECT view E-18

USER_AUDIT_RESOURCE view E-19

USER_AUDIT_TRAIL view E-19

USER_CATALOG view E-20

USER_CLUSTERS view A-3, E-20

USER_CLU_COLUMNS view E-21

USER_COL_COMMENTS view E-21

USER_COL_GRANTS view E-21

USER_COL_GRANTS_MADE view E-21

USER_COL_GRANTS_REC'D view E-22

USER_CROSS_REFS view E-22

USER_DB_LINKS view 22-9, E-22

USER_DUMP_DEST parameter D-26

USER_EXTENTS view E-22

USER_FREE_SPACE E-22

USER_INDEXES view A-3, E-23

USER_IND_COLUMNS E-23

USER_OBJECTS view A-3, E-23

USER_SEGMENTS view 20-8, E-24

USER_SEQUENCES view 5-23, E-24

USER_SESSIONS parameter D-26

USER_SYNONYMS view A-3, E-24

USER_TAB_COLUMNS view A-3

USER_TABLES view A-3, E-24

USER_TABLESPACES view E-25

USER_TAB_AUDIT_OPTS view E-25

USER_TAB_COLUMNS view E-26

USER_TAB_COMMENTS view E-26

USER_TAB_GRANTS view E-26

USER_TAB_GRANTS_MADE view E-27

USER_TAB_GRANTS_REC view E-27
USER_TS_QUOTAS view E-28
USER_USERS view 7-5, A-3, E-28
USER_VIEWS view E-28
USERENV function 18-8
Usernames 17-2, B-28
 distinct within database 17-3
 reusing 17-3
 steps for adding 17-8
 validation 2-5
Users 9-2
 access to SGA 8-3
 adding new 13-7, 17-8
 and rollback segments 4-13
 currently connected 12-17, B-5, B-34
 dropping 17-9
 maximum number of 20-3
 monitoring B-9
 quotas fpr 16-9
 tablespace for temporary segments 16-17, 20-7
 tablespaces for object creation 20-7
Utilities, ORACLE
 shared code or non-shared 8-2

V

VALIDATE INDEX statement 5-18
Validating usernames 2-5
VARCHAR datatype 6-1 - 6-2, 6-7
 description of 6-2
VARGRAPHIC datatype 6-7
Variable header (data block) 5-3
Variable parameters 14-9, 20-4
 effect on SGA 8-3
Variables
 declaring 1-6
Variation, data 5-14
Version numbers
 specific to operating systems 1-8
Versions (ORACLE RDBMS)
 upgrading between 1-11
Views
 auditing in effect 18-6
 data at remote nodes 22-6
 deletion of base tables 5-10
 dependent on terminals 18-8

DML statement restrictions 18-3
locking base tables of 12-12
number of columns 5-10
privilege to create 17-3
privileges relevant to 18-3
purpose of 5-10
revoking privileges 18-4
sequences forbidden in 5-25
storage 5-10
storing definitions 6-5
to restrict table access 18-2
translating into base tables 10-6
V6 compared to V5 A-6
vs. tables in SQL statements 5-10
Virtual memory 8-2

W

Waiting for data 12-2
Waiting for locks 12-12
Week numbering F-2
 ISO standard F-5
WHERE clause 5-25, 6-9
 absence of precludes use of index 19-6
 column order 5-16
 indexed values 5-13
 LONG data in 6-6
 multiple 19-10
 referencing leading portion of index 19-7
 vs. HAVING clause 19-14
Width, column
 implied 5-2
 maximum 5-2
WITH CHECK OPTION 5-10
Writing to the database 15-5

X

X locks 12-13

Z

Zero

internal representation of 6-4

versus null 6-7

Reader's Comment Form

ORACLE RDBMS Database Administrator's Guide
Part No. 3601-V6.0 (Revised April 1989)

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

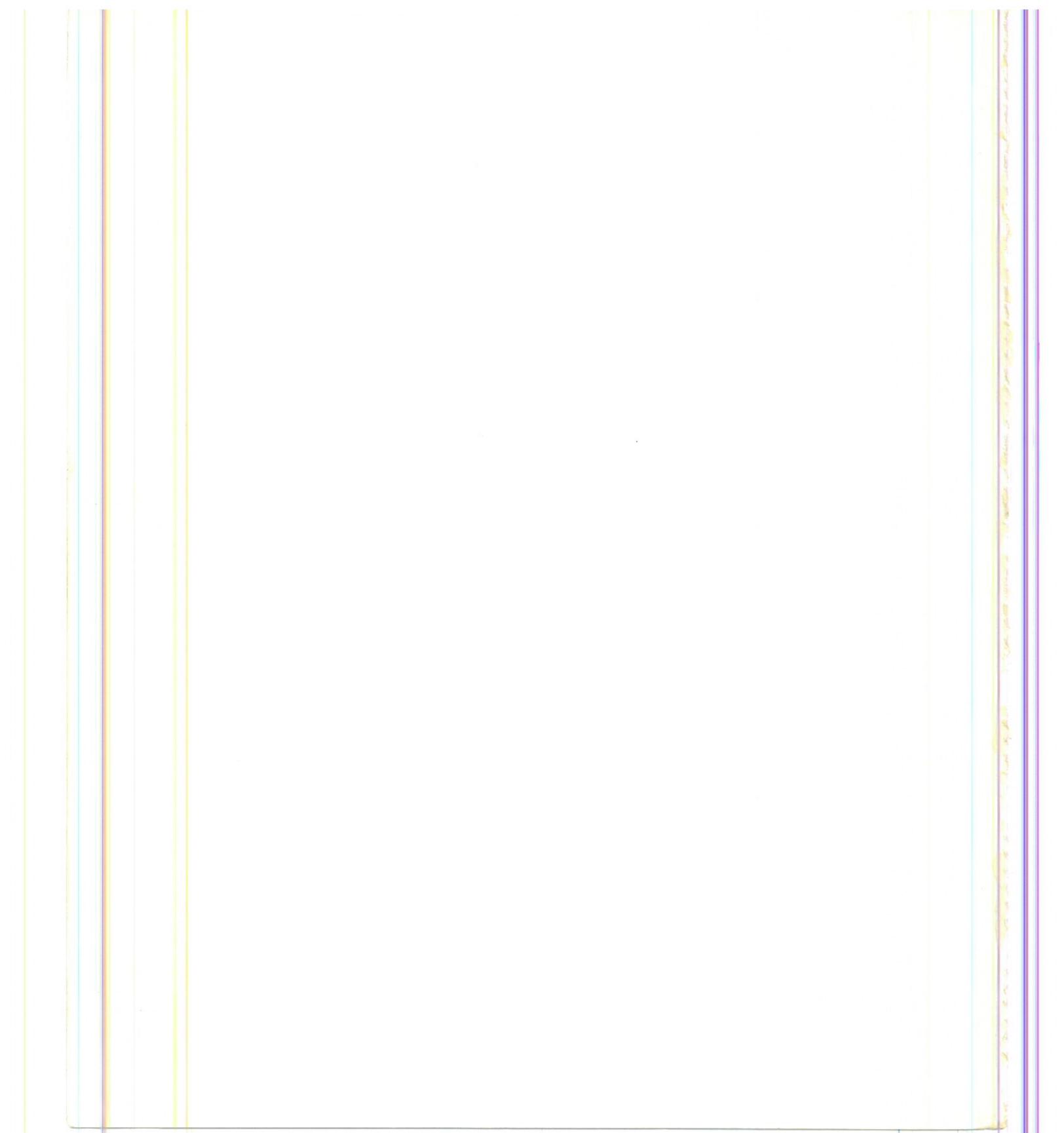
If you find any errors or have any suggestions for improvement, please indicate the topic, chapter, and page number below:

Please send your comments to:

ORACLE RDBMS Product Manager
Oracle Corporation
20 Davis Drive
Belmont, CA 94002
(415) 598-8000

If you would like a reply, please give your name, address, and telephone number below:

Thank you for helping us improve our documentation.



ORACLE®

3601-V6.0
0489